



轻松学

Java Web 开发



电子工业出版社

```

    print("second child,
        %d", getpid());

[root@yux ~]#
[tykoot]# service xinetd restart
*
  xinetd.service: open (socket mode) failed:
  strerror: No such file or directory
[root@yux ~]# service xinetd restart
*
  xinetd.service: open (socket mode) failed:
  strerror: No such file or directory
[root@yux ~]# service xinetd restart
*
  xinetd.service: open (socket mode) failed:
  strerror: No such file or directory

```

Broadview®
www.broadview.com.cn

· 轻松学开发 ·

图解版

轻松学

Java Web 开发

图解学编程，Java Web 竟然这么简单！

本书特点

- 775 幅教学插图，轻松学习技术
- 208 个典型示例，熟练掌握应用
- 603 分钟视频，体验全新方式
- 44 个课后题目，全面测试能力

随书 DVD

603 分钟全程视频 · 本书源代码 · 电子课件

电子工业出版社
ELECTRONIC INDUSTRIAL PRESS
http://www.eip.cn

第1章 浏览器技术

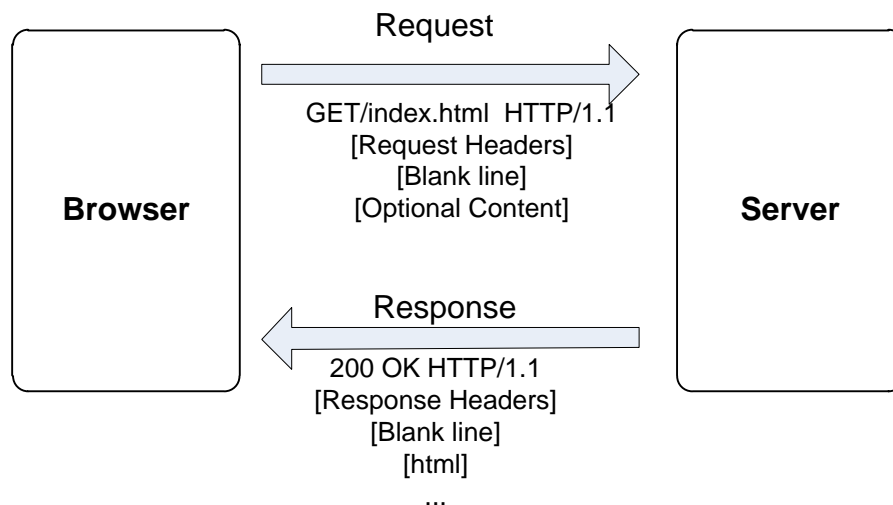
- * 如今随着互联网技术的飞速发展，人们对网络的依赖不断加深。其中，浏览器更是一个连接Internet主要的工具。我们熟悉的浏览器包括微软的IE，Mozilla的FireFox，OPERA等等。那么浏览器真正的作用是什么呢？它是如何工作的呢？官方的解释是这样的定义浏览器的：万维网（Web）服务的客户端浏览程序。可向万维网（Web）服务器发送各种请求，并对从服务器发来的超文本信息和各种多媒体数据格式进行解释、显示和播放。下面我们就来详细地了解它。

1.1 HTTP协议

- * 我们通过浏览器在互联网上浏览新闻，看电影，购物等这些行为看似是顺理成章的事。其实，这一切的行为都是浏览器通过与远在各地的Web服务器进行交互。为了交互的进行，它们需要共同遵守一定的协议来控制。这就是HTTP（Hypertext Transport Protocol），超文本传输协议，一种详细规定了浏览器和web服务器之间互相通信的规则，通过因特网传送万维网文档的数据传送协议。

1.1.1 HTTP协议原理

- * HTTP协议是一种通信协议。它允许将HTML（超文本标记语言）从Web服务器传送到Web浏览器。因此需要Web服务器和Web浏览器都支持该协议。它的具体请求、响应格式如图1.1所示。



1.1.1 HTTP协议原理

- * 当浏览器向web服务器发送一个请求，Web服务器在接受到这个请求后，会返回一个响应给浏览器。这个请求包含一个请求页面的名字和请求页面的信息等。返回的响应包含被请求的页面和被请求页面的信息以及服务器的一些信息等。从图1.1我们也可以看到，浏览器发送这个请求的时候，依据的是HTTP/1.1的格式，因此在返回响应的时候，服务器也必须按照HTTP/1.1的协议格式来响应。

1.1.2 HTTP请求格式

- * HTTP协议对浏览器所发出的request格式有如下三部分的规定：
- * 第一部分是request line。包裹请求的方法、所请求资源的名字以及现在所使用的协议。
- * 第二部分是request headers。它包含浏览器的一些信息。
- * 第三部分是body。其中request headers与body之间有个空行。
- * 具体结构如图1.2所示。

METHOD/path-to-resource HTTP/version-number
User-agent 浏览器的类型 Accept 浏览器接受的MIME类型 Accept language 用户选择的接受语言 Accept-charset 用户首选的编码字符集
Optional request body

- * 其中，METHOD表示请求的方法，如“POST”、“GET”。path-to-resource表示请求的资源。HTTP/version-number表示HTTP协议的版本号。

1.1.3 HTTP响应格式

- * HTTP协议对Web服务器所返回的response也有具体的格式规定。和request一样，response也分为三部分。
- * 第一部分是response line。它包含HTTP协议的版本信息，响应状态等。
- * 第二部分是response header。它包括服务器的一些基本信息。
- * 第三部分是body。response header与body之间也有个空行。
- * 具体结构如图1.4所示。

HTTP/version-number statuscode message
Server 服务器的类型信息 Content-type 响应的MIME类型信息 Content-length 被包含在响应类型中的字符数量
Optional response body

- * 其中，HTTP/version-number表示HTTP协议的版本号。statuscode表示服务器返回的状态码。message表示服务器返回的状态消息。
- * 注意：这里返回的状态码是200，状态信息是OK。表示服务器响应成功，请求被成功的完成，所请求的资源被发送到客户端。

1.1.4 Content type

- * 服务器在接受到请求后，必须能识别要发送的信息类型，比如图片、txt文本、excel表格还是其他的形式。还需要知道网页的编码方式是什么。因此，Content type就是用于定义网络文件的类型以及网页字符的编码。用于决定浏览器以什么形式、什么编码读取这个文件。

1. MIME 类型

- * MIME (Multipurpose Internet Mail Extensions) , 即多功能Internet邮件扩充服务。它是一种多用途网际邮件扩充协议, 服务器会通过这种手段来告诉浏览器它所发送的这些多媒体数据是什么类型的, 需要用何种程序来打开这种文件。最常用的MIME类型有如表1.1所示。

名称	MIME 类型
超文本标记语言	text/html
普通文本	text/plain
Microsoft Word	application/msword
PDF 文档	application/pdf
AVI 文件	video/x-msvideo

2. Content-charset

- * 字符的编码方式有很多种。有的支持中文显示，有的支持英文显示。其中最常见的字符集编码类型如表1.2所示。

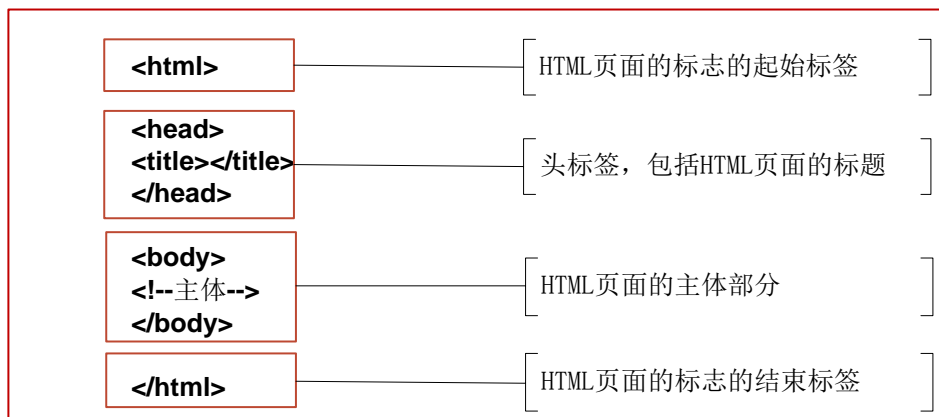
charset 类型	字符集编码类型
ISO-8859-1	拉丁语系1
Big5	繁体中文
UTF-8	通用子集转化格式（8位）
ISO-2022-JP	日语
ISO-2022-KR	韩国语
GBK	简体中文（兼容GB2312）
GB2312	汉字国标码

1.2 HTML

- * HTML (Hypertext Markup Language) , 即超文本标记语言, 是用于描述网页文档的一种标记语言。它是一种规范, 一种标准, 通过标记符号来标记要显示网页的各个部分。任何动态语言都离不开HTML的支持。所以在学习Web开发之前, 读者首先要掌握HTML的相关基础知识。

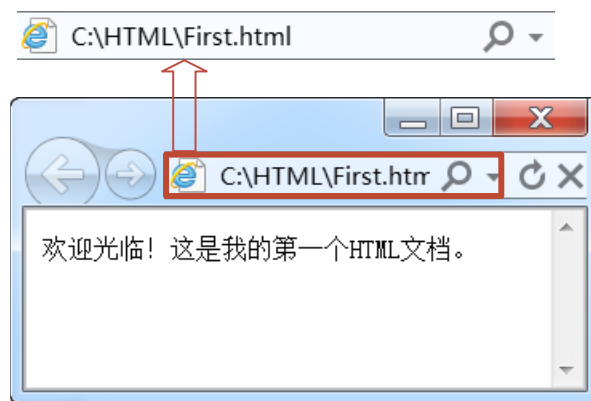
1.2.1 标记语言

- * 标记语言，也叫做置标语言。是将文本以及文本相关的其他信息结合起来，展现出文档的结构和数据处理细节的电脑文字编码。
- * 标记语言根据使用范围的不同有很多类型。例如用户网页信息的是超文本标记语言（HTML），用户电脑处理各种信息的是可扩展的标记语言（XML）等。由于类型的不同，它们各自的语法也各不相同。下面我们重点介绍的是用于搭建网页的超文本标记语言，即HTML。HTML的大致结构如图1.6所示。



1.2.1 标记语言

- * 我们在地址栏中输入文件的存储地址，就可以运行这个html文件，运行结果如图1.8所示。



- * 注意：一般情况下，可以包括其他内容的HTML标签都是成对出现的，例如上面例子中的<title></title>这对标签，它包含了一个文字的标题信息，所以成对出现。而
这样的标签仅仅是一个回车换行的作用，它不包含其他内容，所以不成对出现。

1.2.2 超链接

- * 在HTML中实现页面的跳转的主要方式是用超链接的形式。超链接的用途非常广泛，可以是一个字、一个词，或者一组词，也可以是一幅图像。用户可以点击这些内容来跳转到新的文档或者当前文档中的某个部分。例如我们打开百度首页，如图1.7所示。它的每个词汇单击后都可以链接到一个新的页面，甚至在点击首页中图片的时候也打开了一个新的页面，这都是超链接的应用。
- * 在HTML中超链接的标签是<a>，它的语法格式如图1.8所示：

<code><a</code>	<code>href=" url"</code>	<code>target=" _blank"</code>	<code>></code>	<code>link words</code>	<code></code>
					
	链接的相对路径	链接后的显示方式		链接文字（图片）	

1.2.3 静态页面

- * 当一个网页页面仅仅由HTML语言代码组织起来的，那么这个页面就是一个静态的页面。它不会与数据库、服务器进行交互，只能通过浏览器进行显示，是一个独立存在的文件。现今我们通过浏览器浏览Internet的时候，所看到的大部分网站都不是静态网站。因为它的显示单一，功能简单，已经被动态网站所取代。
- * 与静态网站相对应的是动态网站，所谓的动态网站，就是用户可以通过浏览器与服务器端进行数据交互的网站。
- * 动态网站不只是HTML语言组织起来的。它是由HTML语言与其他的脚本语言共同组织起来的，例如HTML+ASP、HTML+PHP、HTML+JSP等。HTML语言起到一个显示的作用，至于交互的动作及过程，都是由脚本语言完成的。有关动态网站的内容我们会在后面章节中为大家讲解。

1.3 HTML常用表单标签

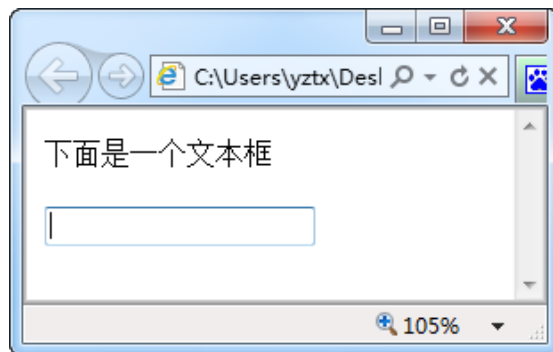
- * 在HTML的布局标签中，表单标签是使用频率最高的一个。它可以把一组信息用表格的形式表示出来。这一节我们就来看表单元素是如何实现的。

1.3.1 表单元素

- * 在HTML的标签库中有<form></form>这样一个成对标签，它用于向一个目标地址提交一些数据。在这个标签中，我们可以设计文本框、单选框、复选框和按钮等等一些元素用于获取数据，这些元素都称为表单元素。下面我们介绍几个最常用的表单元素。

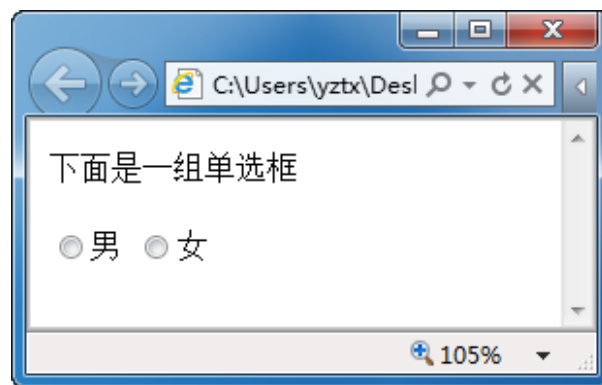
1. 文本框

- * 在HTML页面最常用的就是一个单行的文本框了。
它的语法格式是：
- * `<input type="text" name="text">`
- * 在浏览器页面显示的效果如图1.11所示。



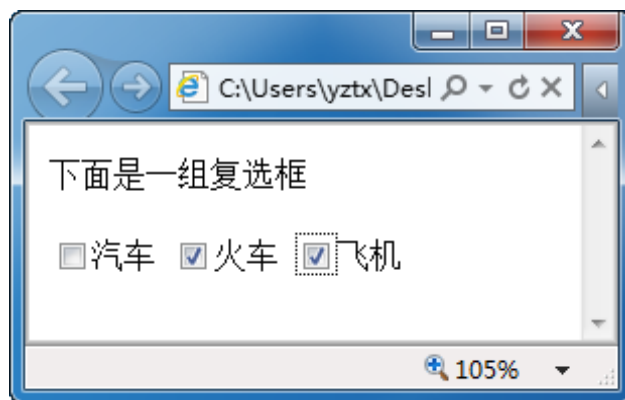
2. 单选框

- * 单选框就是在并列的几个值中只能选择一项。对于一组元素，必须保证它们的name属性值相同、语法格式如下所示：
- * `<input type="radio" value="值1" name="dxk">`
- * `<input type="radio" value="值2" name="dxk">`
- * 在浏览器上显示的效果如图1.12所示。



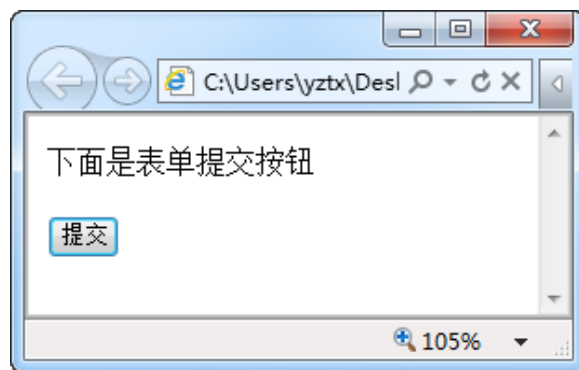
3. 复选框

- * 当用户需要从若干给定的选择中选取一个或若干选项时，就会用到复选框。对于同一组值，必须要保证它们的name值相同。语法格式如下：
- * `<input type="checkbox" value="值1" name="dxk">`
- * `<input type="checkbox" value="值2" name="dxk">`
- * 在浏览器上显示的效果如图1.13所示。



4. 提交按钮

- * 当表单要提交时，单击这个按钮，表单就会将表单中所选数据提交到目的地址。语法格式如下：
- * `<input type="submit" value="提交">`
- * 在浏览器上显示的效果如图1.14所示：



- * 除了上述介绍的几个元素外，表单还有其他很多的元素，例如重置按钮、密码框和下拉列表等等。

1.3.2 表单元素的属性

- * 在这些元素中，有一些共同的属性，即type、name和value。那么这三个属性都有什么作用呢？
- * type属性：表示该元素的类型。有text（文本框）、checkbox（单选框）和button（按钮）等值。
- * name属性：表示该元素的名称，只能有唯一值。
- * value属性：设置该元素的默认值。
- * 与提交表单之前相比，字符串后面多了“？”
hobby=test&tijiao=botton”，这表示，在我们提交表单后，表单会将表单元素中文本框和按钮属性的值以“key=value”的形式附加到地址字符串的后面，之间用符号“&”隔开。

1.3.3 表单中添加目的地址

- * 在示例1.4中我们可以看到，在form表单中并没有目的地址属性。表单提交仅仅是包含着数据但并没有提交到某个页面。在这里我们在form中添加action这个属性。这个属性的作用就是表示表单要提交的目的地址。语法格式如图1.17所示：

<form action= “url” > </form>

表单提交地址的相对路径

1.3.4 表单中添加数据的提交方式

- * 当我们的数据可以提交到目的地址后，疑问也随之产生，就是数据时以何种方式提交的呢？是通过浏览器的地址栏中的附加到地址字符串方式提交的吗？还是有其他方式？
- * form表单中还有一个属性method。它表示表单中数据的提交方式。它有两个可选值：POST和GET。
- * POST：将数据打包，以隐含的方式传递。
- * GET：附加到URL上，通过URL来传递数据。

1.4 CSS基础简介

- * 在前面的内容中讲解了HTML，现在我们已经基本可以编出最简单的网页了，但是仅仅这样还是不够的，一个专业的网页需要在字体、颜色、布局等方面进行各种设置，需要给用户带来视觉的冲击。这一节我们简要介绍一下页面美化技术——CSS。

1.4.1 CSS属性设置

- * CSS (Cascading Style Sheets) 即层叠样式表，也就是通常所说样式表。CSS是一种美化网页的技术。通过使用CSS，可以方便、灵活地设置网页中不同元素的外观属性，通过这些设置可以使网页在外观上达到一个更高的级别。CSS美化网页就是通过设置页面元素的属性来实现的，下面我们就来介绍CSS属性设置的一些基本方法。

1. 字体、颜色、背景等属性设置

- * 字体、颜色、背景属性是CSS最基本的属性，其最常用的属性方法如表1.3所示。

属性方法	方法作用
font-family	定义使用哪种字体
font-style	定义是否使用斜体
font-weight	定义字体的粗细
font-size	定义字体的大小
background-color	定义背景颜色
background-image	定义背景图片
background-attachment	定义滚动
background-position	定义背景图片的初始位置

2. 鼠标样式属性设置

- * 在一些网页中，我们经常会遇到这样一种情况，当把鼠标移到不同区域，或者是在执行不同功能的时候，鼠标的形状都会发生变化。这种功能的实现其实非常简单，就是控制CSS中的cursor属性来实现的。

1.4.2 CSS绝对定位

- * 在HTML中布局一般情况下需要使用表格，如果要定位只有通过表格的嵌套来实现，这种方法的确可以在一定程度上解决问题，但是却不能随意定位页面元素，而且对某个元素位置的改变有可能影响到整个页面的布局。在CSS中提供了灵活的定位的方法，所以在页面布局中我们又多了一种可以选择的方案。
- * CSS中常用的定位属性如表1.4所示。

属性	作用	属性	作用
absolute	绝对定位	width	定义宽
relative	相对定位	height	定义高
static	静态定位	overflow	定义内容超出的处理方法
left	定义横坐标	z-index	定义立体效果
top	定义纵坐标	visibility	定义可见性

1.4.3 CSS实现表格变色

- * 在一些Web应用中间经常会用表格来展示数据，当表格行数比较多的时候，就容易后看错行的情况发生，所以需要一种方法来解决这个问题。在这里我们采取这样一种措施，当鼠标移到某一行时，这行的背景颜色发生变化，这样当前行就会比较突出，不容易出错。

1.5 小结

- * 本章主要讲解了与浏览器相关的技术，包括HTTP协议、HTML、HTML中用于提交数据的表单以及CSS技术的基础知识。重点是在理解HTTP的请求与响应的交互原理以及表单中各个元素的使用法。难点在于HTML与CSS技术的熟练运用。在接下来的章节中，我们将开始JSP、Servlet等关键技术的讲解，这也是本书的重点。

第2章 JSP基础

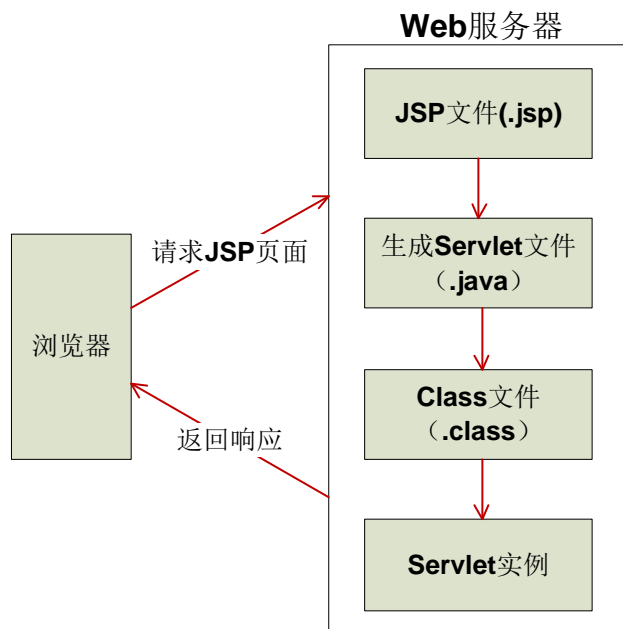
- * 上一章我们主要讲解了从浏览器端用表单提交数据，数据通过POST或者GET的方式提交到服务器端。那么在服务器端是通过什么技术来获取这些数据呢？并且对这些数据是进行怎么处理的呢？接下来在本章中，我们主要讲解的是在服务器端获取和处理数据的技术——JSP（Java Server Pages）。

2.1 JSP与服务器

- * 客户端通过表单将数据提交到action指定的目的地址。在这个目的地址指向的页面，需要将数据提取出来。这就需要有一个动作脚本来完成动态网页技术中的数据交互。这种动作脚本与HTML语言相结合来获取和处理表单提交的数据。在Java Web中，这种用于服务器端处理数据的动作脚本就是JSP。
- * JSP是Java Server Pages的简称，它是在传统的HTML文件中插入Java程序段和JSP标记，形成的JSP (.jsp) 文件。它是一种动态网页技术，遵从动态网页的技术标准。

2.1.1 JSP在服务器上工作原理

- * JSP文件是运行在服务器端的脚本文件，它由HTML语言、Java代码和一些独特的JSP标记组成。由于它包含了Java程序段，所示它需要被服务器编译才能运行。我们知道JSP页面被部署在Web服务器或应用服务器上。整个JSP工作机制如图2.1所示。



2.1.1 JSP在服务器上工作原理

- * 服务器管理JSP页面分为两个阶段：转换阶段和执行阶段。
- * (1) 当有一个JSP请求到来时，服务器会首先检验JSP页面的语法是否正确，将JSP转换成Servlet（Servlet就是用Java语言实现的CGI程序，后面章节将详细介绍）源文件，然后调用javac工具类编译Servlet源文件生成.class文件，这就是转化阶段。
- * (2) Servlet容器加载转化后的Servlet类，实例化一个对象处理客户端的请求。在请求处理完成后，响应对象被服务器接受，服务器将HTML的格式的响应信息发送给客户端，这一阶段便是执行阶段。
- * JSP页面的第一次执行要花费一些时间，去完成JSP页面到Servlet的转换。当再次请求时，JSP服务器就会直接执行第一次请求时产生的Servlet，而不再进行JSP文件的转换。

2.1.2 Web服务器Tomcat的搭建

- * JSP页面必须被部署和运行于Web服务器中，所谓的Web服务器是指为特定组件提供服务的一个标准化的运行时的环境，其中封装了JSP运行所需要的底层API，为组件提供事务处理，数据访问，安全性，持久性等服务。
- * Web的服务器有很多种，其中Tomcat就是一个免费并且开源的JSP服务器。它是Apache软件基金会的Jakarta项目中的一个核心项目。它由Apache、Sun和其他一些公司及个人共同开发而成。目前它是使用最广泛的JSP服务器。到目前为止，它的最新版本是Tomcat 7.0。读者可以从Apache Tomcat官方网站 (<http://tomcat.apache.org/>) 上选择下载Tomcat 7.0。

2.1.2 Web服务器Tomcat的搭建

- * 注意：Tomcat 7.0.29是笔者写作时的最新版本。读者下载时候，可能是更新的版本。但是只要大版本号一致，就不会影响大家学习。
- * 在单击Download之后，我们进入具体的版本选择页面，在这里我们选择“32-bit/64-bit Windows Service Installer”版本进行下载。
- * 下载完成后，双击下载的Tomcat安装文件apache-tomcat-7.0.29.exe，弹出安装对话框。

2.1.2 Web服务器Tomcat的搭建

- * 在单击Next按钮之后，我们要对Tomcat的路径进行配置。首先选择读者计算机中所安装Java的jre地址，然后设置Tomcat的安装路径。为了方便将来在Tomcat中部署应用程序，我们建议读者选择比较浅的路径进行安装，比如“D:\Tomcat”。
- * 单击Install按钮，完成Tomcat的安装。
- * 安装完成后，会在系统栏中加载一个绿色的服务器图标。

2.1.2 Web服务器Tomcat的搭建

- * 在Tomcat上发布Web应用之前，我们首先先了解一下Tomcat的目录结构。打开Tomcat的安装路径后，如图2.8所示。

名称	修改日期	类型	大小
bin	2012/7/31 10:26	文件夹	
conf	2012/8/2 15:49	文件夹	
lib	2012/7/31 10:26	文件夹	
logs	2012/8/2 15:49	文件夹	
temp	2012/7/31 10:26	文件夹	
webapps	2012/7/31 14:23	文件夹	
work	2012/7/31 10:29	文件夹	
LICENSE	2012/7/3 18:33	文件	53 字节
NOTICE	2012/7/3 18:33	文件	2 KB
tomcat	2012/7/3 18:33	图标	22 KB
Uninstall	2012/8/2 15:49	应用程序	66 KB

具体的每个文件夹的描述如下表2.1所示：
表2.1 Tomcat目录描述

目录	描述
/bin	存放在Windows平台以及Linux平台上启动和关闭Tomcat的脚步文件
/conf	存放Tomcat服务器的各种配置文件，其中最重要的配置文件是server.xml
/lib	存放Tomcat服务器所需的各种JAR文件
/temp	存放Tomcat产生的临时文件
/logs	存放Tomcat的日志文件
/webapps	当发布Web应用时，默认情况下把Web应用文件存放于此目录下
/work	Tomcat把由JSP生成的Servlet放于此目录下

2.1.3 安装MyEclipse

- * MyEclipse是目前应用最为广泛的Java应用程序集成开发环境。它是由Genuitec公司开发的一款商业化软件。用户可以通过购买或因特网下载获得其安装包。本书所应用的为最新的MyEclipse 10.0版本。
- * 首先我们双击MyEclipse的安装文件myeclipse-10.0-offline-installer-windows.exe，开始MyEclipse的安装。

2.1.4 MyEclipse 中集成Tomcat服务器

- * 在MyEclipse中其实已经自带了一个Tomcat服务器，但是为了日后我们程序的开发、部署和运行更加方便和快捷，我们将用户安装的Tomcat服务器集成到MyEclipse中。具体的集成步骤如下所示：
 - * (1) 选择MyEclipse菜单栏中的Window|Preferences命令，在弹出的窗口中选择MyEclipse|Servers|Tomcat.
 - * (2) 根据我们安装的Tomcat的版本我们选择Tomcat 7.x链接进行配置，系统出现Tomcat 7.x服务器配置窗口，首先我们将“Tomcat Server”选项设置为“Enable”，然后单击“Tomcat home directory”选项后的“Browse...”按钮，选择Tomcat 7.x的安装目录。
 - * (3) 这样我们就基本完成了MyEclipse中Tomcat的集成，然后我们单击MyEclipse工具栏中的“Run MyEclipse Servers”按钮，将会看到Tomcat 7.x服务器。单击“Start”按钮，启动Tomcat服务器。

2.1.5 MyEclipse 中 JSP 页面的创建

- * 完成了各项软件的配置之后，我们一起来学习如何在 MyEclipse 中进行 JSP 页面的创建。在 MyEclipse 中，JSP 页面是以 Web 项目的形式组织起来的。所以要创建 JSP 页面之前，必须要创建一个 Web 项目，创建的具体步骤如下：
- * (1) 选择 MyEclipse 菜单栏中的 File|New|Project... 命令，将显示项目对话框，在其中选择“Web Project”选项。
- * (2) 单击 Next 按钮，启动创建 Web 项目向导。在 Project Name 文本框中输入项目名称 FirstWeb，然后选择 J2EE Specification Level 下的 Java EE 6.0 单选按钮。最后单击 Finish 按钮完成 Web 项目的创建。
- * 注意：J2EE Specification Level 选择哪个版本取决于读者所应用服务器版本。例如 Tomcat 4.x 以下版本只能选择 J2EE 1.4，而 Tomcat 6.x 服务器就可以选择 Java EE 5.0。我们应用的 Tomcat 7.x，所以建议选择 Java EE 6.0。

2.1.5 MyEclipse 中 JSP 页面的创建

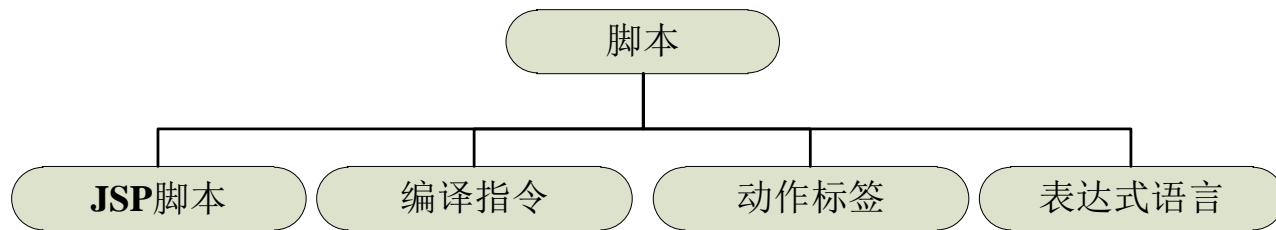
- * Web项目创建完成后，就可以在该项目中创建JSP页面了，具体步骤如下：
- * （1）选择MyEclipse菜单栏中的File|New|JSP(Advanced Templates)命令，将显示JSP创建窗口。我们可以在此创建窗口中确定JSP页面的名称和存储路径。
- * （2）单击Finish按钮，完成JSP页面创建。

2.1.6 MyEclipse 中 Web 项目的发布和运行

- * Web项目在开发完成之后，需要发布到Web服务器上才能够被访问和运行。所以我们必须要掌握如何在MyEclipse中进行Web项目的发布和运行。在开发过程中Web项目的发布和运行的步骤如下所示：
- * （1）在Package Explorer视图，右击FirstWeb项目，在弹出的菜单中选择MyEclipse|Add and Remove Project Deployments...命令，系统出现项目部署对话框。
- * （2）单击Add按钮，出现创建新部署对话框，选择Tomcat 7.x部署到服务器上，并在Deploy type中选择Exploded Archive开发模式。
- * （3）单击Finish按钮，项目将部署到所选择的服务器中。
- * （4）然后我们启动Tomcat服务器，输出的日志就会自动显示在Console视图中，便于读者浏览和判断服务器是否正常启动完毕。

2.2 JSP的基本语法

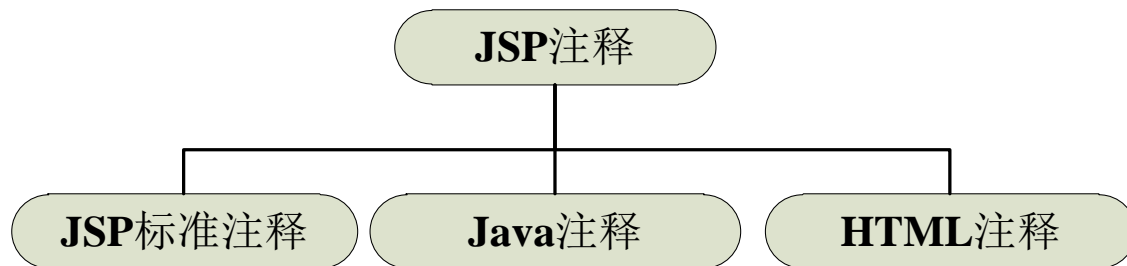
- * JSP网页主要分为脚本和网页数据两部分。网页数据就是JSP服务器不需要处理的部分。例如，HTML的内容会直接送到客户端执行。脚本是必须经由JSP处理的部分，大部分脚本都以XML作为语法基础，其可以分为四种类型：JSP脚本、编译指令、动作标签和表达式语言，如图2.25所示。



- * 这一章我们会为大家讲解前面三种类型，关于表达式语言（Expression Language）我们会在后面的章节中为大家单独讲述。本节我们首先来看JSP脚本。

2.2.1 JSP注释

- * JSP程序中可以包含3种不同类型的注释，如图2.26所示。



1. JSP标准注释

- * JSP标准注释通常用来编写JSP说明文档，当JSP网页在服务器中编译时将被完全忽略，其语法格式如图2.27所示。

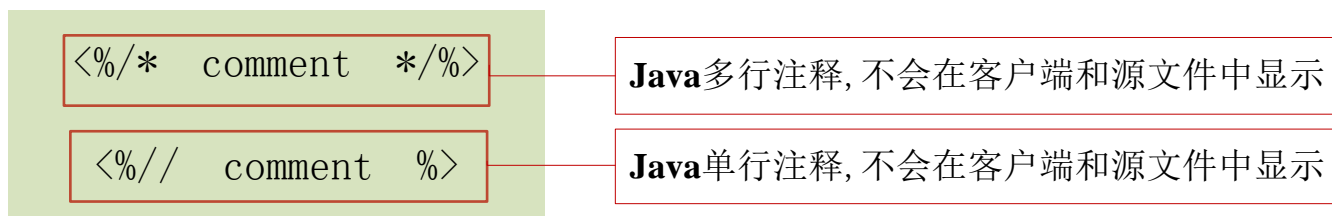
```
<%-- comment --%>
```

以“<%--”开头，以“--%>”结尾，
不会在客户端和源文件中显示

- * 这是开发程序员专用的注释，可以讲开发人员希望隐藏的JSP程序注释起来。这些注释将不会显示在客户的浏览器中，用户也不能通过浏览器的“查看”|“源文件”操作，在源代码中查到。

2. Java注释

- * 在JSP的Java程序中，我们也可以遵循Java语言本身的注释规则，即在一对“<%”和“%>”中，将Java注释添加进去，如图2.28所示。



- * Java注释在JSP页面编译时，也将会被完全忽略，同样用户也不能通过浏览器的“查看”|“源文件”操作，在源代码中查到。

3. HTML 注释

- * HTML注释是一种能在客户端显示的注释，它的语法规则如图2.29所示。

```
<!-- comment [<%= expression %>] -->
```

以“<!--”开头，以“-->”结尾，可以在客户端和源文件中显示，注释中可以使用JSP动态表达式

- * JSP页面中的HTML注释和HTML中的注释很相像，也可以通过浏览器的“查看”|“源文件”操作，在源代码中查到。唯一不同的是，可以在这个注释中使用JSP表达式，从而记录一些JSP页面动态运行结果。
- * 由于这是我们的第一个JSP文件，所以我们为大家讲解一下JSP文件的运行过程。首先在MyEclipse中建立创建一个名为HelloWorld的JSP文件。

3. HTML 注释

- * 然后在文本编辑区内输入图2.30所示的内容。
- * 接着对HelloWorld.jsp文件进行编译，编译方式有三种：我们可以执行菜单栏中的Run|Run命令进行编译，也可以使用快捷键“ctrl+F11”，或者直接点击工具栏上的编译按钮。
- * 编译通过后，读者需要在自己的浏览器中输入文件的运行地址才能将JSP文件的运行结果显示出来。
- * 我们可以右击浏览器界面，在弹出菜单中单击“查看源文件”选项，就可以看出三种注释方法的区别。

2.2.2 声明变量和方法

- * 声明用于声明JSP程序中要用到的一个或多个变量和方法。在JSP中声明变量和方法，是以“<%!”开头，以“%>”结尾的，多个变量和方法以“;”分隔。JSP声明的语法如图2.35所示。

```
<%! declaration;[declaration;]…… %>
```

以“<%!”开头，以“%>”结尾，中间的声明用“;”隔开

2.2.3 JSP表达式

- * JSP表达式用来在JSP页面中输出作为运行结果的字符串或是数值变量。JSP表达式可以被看做是一种简单的输出形式，任何在Java语言规范中有效的表达式都能够作为JSP表达式在JSP页面中使用。JSP表达式语法如图2.37所示。

`<%= expression %>`

以“<%=”开头，以“%>”结尾，中间包含一段合法的表达式

- * 表达式具体的使用示例代码如下：
- * `<font size=<%=i%>>世界，你好！ `
- * `<%=circle.getArea() %>`
- * 由于表达式的书写格式比较繁琐，而且完全可以由JSP中的内置对象out（在后面章节中会作介绍）来替代，因此在实际开发中，JSP表达式很少被用到。

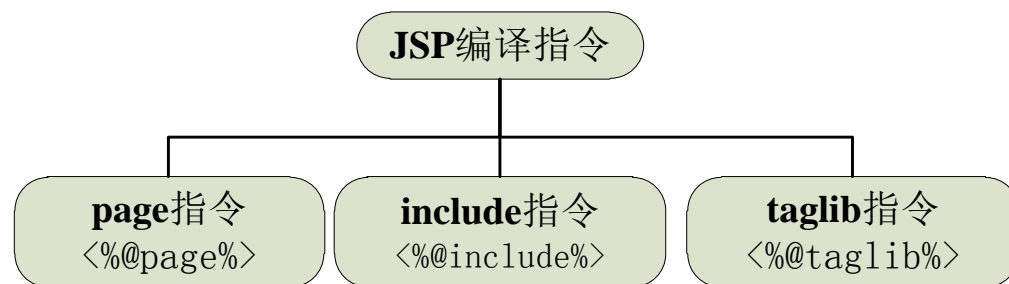
2.3 JSP编译指令

- * 编译指令是指在JSP文件中包含在符号“<%@”和符号“%>”之间的部分。它不向客户端输出任何内容，是用来设置全局变量、声明类、方法和输出内容的类型的指令。编译指令元素的格式如图2.38所示。

```
<%@ directive {attribute=" value" , .....} %>
```

以“<%@”开头，以“%>”结尾，中间**directive**代表不同的指令

- * 在JSP中的编译指令包括3种指令，如图2.39所示。



2.3.1 page指令

- * 页面指令用来定义JSP文件中的全局属性，这些全局属性都是影响整个页面的重要属性。一个JSP文件中可以有多个页面指令，在JSP文件被解析为Java代码时，这些页面指令也被解析为对应的Java代码。
- * page指令可以在一个JSP文件中多次、多处使用，但是其中的属性却只能使用一次（import除外），重复的属性设置将会覆盖掉先前的设置。无论用户将page指令放在JSP程序的任何地方，它的作用范围都是整个JSP页面。

2.3.1 page指令

- * 图2.40的语法列出来了大部分页面指令属性及其取值，下面通过表2.1去详细解释每个属性的意义和使用方法。

属性	描述	默认值	例子
language	指定使用的脚本语言，目前只能为“Java”	“Java”	language="Java"
import	和Java代码中import的作用一样，如果有多个包，用“,”隔开	默认忽略该属性	import="Java.io.*,Java.utilis.*"
session	这个页面是否参与一个指定的HTTP会话	true	session="true"
buffer	指定客户端输出流的缓冲模式，可以为一个数值或none	不小于8kb	buffer="32kb"
autoFlush	设置当缓冲区满时，是否实现自动刷新	true	autoFlush="true"
Info	指定关于该JSP文件的信息	默认忽略该属性	info="第一个JSP页面"
isErrorPage	表明当前页是否为其他页的errorPage，由此决定当前页是否可以使用exception对象	false	isErrorPage="false"
errorPage	定义当前页运行出现异常时调用的页面	默认忽略该属性	errorPage="error.jsp"
isThreadSafe	指定该JSP文件是否能多线程使用，由此判断是否可以处理多个用户的请求	true	isThreadSafe="true"
contentType	定义JSP字符编码和页面响应的MIME类型	TYPE=text/html CHARSET=iso8859-1	contentType="text/html;charset=UTF-8"
pageEncoding	指定该JSP文件的字符编码	pageEncoding="iso-8859-1"	pageEncoding="UTF-8"
isELIgnored	指定EL（表达式语言）是否被忽略，如果为true，则容器忽略“\${}”表达式的计算	Servlet2.3之前的版本忽略该属性	isELIgnored="true"

2.3.2 include指令

- * include指令用来将指定位置上的资源包含在当前JSP文件中。在JSP文件被编译为Java文件时，这些被包含的资源会被作为JSP文件的一部分被翻译为Java文件。所以这些资源可以看作是JSP文件的一部分。在JSP文件中include指令的语法形式如图2.43所示。

```
<%@ include file= "filename" %>
```

以“<%@”开头，以“%>”结尾，中间为所包含的文件名

- * 图中的filename指定要包含的资源的文件名。如果filename以“/”开头，那么该文件的路径是参照JSP应用的上下文路径；如果filename是以文件名或目录名开头，那么该文件的路径就是当前JSP文件的路径。

2.3.2 include指令

- * include指令的具体示例代码如下：
- * `<% @ include file= "HelloWorld.jsp" %>`
- * `<% @ include file= "name.html" %>`
- * `<% @ include file= "/FirstWeb/test.txt" %>`
- * 其中 `"/FirstWeb/test.txt"` 表示所要包含的文件test.txt在当前使用的JSP文件同级路径中的FirstWeb目录中。
- * 值得注意的是，被包含JSP文件中不能还有page指令，以免同JSP文件中已有的同样指令发生冲突。

2.3.3 taglib指令

- * taglib指令是当JSP页面中引用了用户自定义标签时，用来声明这些用户自定义的标签的。JSP引擎使用taglib指令以确定在JSP中遇到用户自定义标签时应该怎样去做。taglib指令的语法形式如图2.46所示。

```
<%@ taglib uri="taglibURL" prefix="tagPrefix" %>
```

唯一确定标签库的位置

定义标签中的前缀字符串

2.4 JSP动作指令

- * JSP动作是一种特殊的标签。利用XML语法格式的标签来控制JSP引擎的行为，影响JSP运行时的功能，并返回客户端的响应。JSP动作都以“<jsp:”开头，相对应地则以“/>”结束。
- * JSP主要包括的动作如图2.49所示。

<jsp:include>	用于在客户端请求时间内把静态或者动态的资源包含在JSP页面内
<jsp:forward>	用来将请求转发到其他的JSP页面、Servlet或者静态资源文件
<jsp:param>	用来给其他的标签提供参数
<jsp:useBean>	用来在JSP文件中使用一个JavaBean的实例，并声明该实例的名字以及作用范围
<jsp:setProperty>	用来给JavaBean设置属性，该标签会调用JavaBean的setXXX()方法去完成一个或者多个属性的设置
<jsp:getProperty>	用来获取JavaBean中属性的值。它将JavaBean的值转换为String类型，然后发送到输出流中

2.4.1 <jsp:include>动作指令

- * <jsp:include>动作元素用于在客户端请求时间内把静态或者动态的资源包含在JSP页面内。包含的静态或动态的资源在page属性中用URL的形式指定。<jsp:include>指令的格式如图2.50所示。

```
<jsp:include page="fileName" flush="true"/>
```

空标签形式

```
<jsp:include page="fileName" flush="true">  
<jsp:param name="paramName" value="paramValue"/>  
...  
</jsp:include>
```

体标签形式，
若在包含资源
的同时还要传
递参数，则使
用体标签形式

- * include动作指令可以在JSP页面中动态包含一个文件，这与include指令不同，前者可以动态包含一个文件，文件的内容可以是静态的文件也可以是动态的脚本，而且当包含的动态文件被修改的时候JSP引擎可以动态对其进行编译更新。而include指令仅仅是把一个文件简单的包含在一个JSP页面中，从而组合成一个文件，仅仅是简答的组合的作用。

2.4.2 <jsp:forward>动作指令

- * forward动作指令可以用来控制网页的重定向。即将请求转发到其他的JSP页面、HTML页面或者Servlet类。<jsp:forward>动作指令的语法格式如图2.54所示。

```
<jsp:forward page="url"/>
```

空标签形式

```
<jsp:forward page="url">  
<jsp:param name="paramName" value="paramValue"/>  
...  
</jsp:forward>
```

体标签形式，
若在包含资源
的同时还要传
递参数，则使
用体标签形式

- * 只要指明page的值，当JSP执行到这行代码的时候就可以直接跳转到对应的页面。

2.4.3 <jsp:param>动作指令

- * <jsp:param>动作指令主要用于传递参数，为其他动作指令提供附加信息，它必须和<jsp:include>、<jsp:forward>等动作指令一起使用，可以将<jsp:param>动作指令中的值传递到其他动作指令所要加载的文件中去，从而实现向所加载的页面动态传递参数的功能。<jsp:param>动作指令的语法格式如图2.57所示。

```
<jsp:param name="paramName" value="paramValue"/>
```

name表示传递参数名，
value表示传递参数值

- * 通过<jsp:param/>可以给请求的页面传输一个或多个参数。如果要传输多个参数，可以使用多个<jsp:param/>标签。当给一个页面传输参数时，该页面肯定是动态页面。

2.5 小结

- * 本章主要介绍了Tomcat服务器和MyEclipse开发软件的安装过程，以及和JSP有关的基本语法等方面的内容。并通过一个HelloWorld程序对MyEclipse中Java Web应用开发的基本步骤和流程有了一个大体的认识。本章的重点是讲解Java Web程序运行环境的搭建和JSP基本语法，难点是运用JSP编译指令和动作指令的运用。希望读者多加练习，以为后面章节中JSP学习的深入打好基础。

第3章 JSP 内置对象

- * JSP内置对象是为了简化JSP页面开发而建立的一些内部对象。这些对象不需要声明，可以在程序中直接使用。它们是JSP语言的精髓，掌握常见内建对象的使用技巧是进行JavaWeb开发必不可少的。正确地掌握和灵活地使用JSP内置对象是学习JSP开发的重中之重。本章就来为大家讲解九大内置对象，如表3.1所示。

内置对象	主要作用
request	包含客户端请求信息
response	页面传回给用户端的相应信息
out	用来向客户端浏览器输出信息的数据流
session	为发送请求的客户建立会话
application	保存整个应用程序的共享信息
pageContext	保存当前JSP页面的共享信息
config	读取初始化参数
page	代表JSP网页本身
exception	获取运行时的异常

第3章 JSP 内置对象

- * 由于我们在实际开发中经常会遇到中文乱码问题，所以在本章的结尾我们会为大家单独用一小节来讨论如何解决中文乱码的问题。

3.1 request 内置对象

- * request对象用来接收客户端提交的各种信息。如果要与用户的互动，必须要知道用户的需求，然后根据这个需求生成用户期望看到的结果。这样才能实现与用户的互动。在Web应用中，用户的需求就抽象成一个request对象，这个对象中间包括用户所有的请求数据，例如通过表单提交的表单数据等方式传递的参数，这些就是用户的需求。表3.1中列举了request对象中的常用方法及方法描述。

方法	方法描述
getParameter(String name)	获取客户端传给服务器的参数值，name指定表单中参数的名字
getParameterNames()	获取客户端传给服务器的所有参数的名字，返回的结果是一个枚举实例
getParameterValues(String name)	获得某一个参数的所有的值，name指定参数名字
getAttribute(String name)	获得request对象中某一个属性的值，name为属性名，如果该属性不存在，则返回null
setAttribute(String name,Java.lang.Object.objt)	给request对象设置一个名字为name的属性值，该值有objt设置
removeAttribute(String name)	移除request对象中名字为name的属性
getAttributeNames()	返回request对象中所有属性的名字，结果是一个枚举类型
getCookies()	返回客户端所有的Cookie对象，结果是一个Cookie数组
getCharacterEncoding()	返回客户端请求中字符的编码方式
getContentLength	返回客户端请求的body的长度
getMethod()	返回客户端向服务器传输数据的方法，如get、post、header、trace等
getRequestURL()	获取发送请求的客户端地址
getRemoteAddr()	获取客户端的IP地址
getServerName()	获取服务器的名字
getServerPort()	获取服务器的端口号
getServletPath()	获取客户端所请求的脚本文件的文件路径

3.1.1 获取用户提交的表单信息

- * request对象最主要的一个作用就是用来封装用户提交的表单信息，然后通过如下两个方法来获取用户提交的表单信息。
- * `getParameter(String name)`: 获取客户端传给服务器的参数值。
- * `getParameterValues(String name)`: 获得某一个参数的所有的值。

3.1.1 获取用户提交的表单信息

- * 这是一个典型的获取用户表单的示例，它体现了JavaWeb中数据和提交的大致实现过程，如图3.5所示。

客户端输入

```
用户名: <input type="text" name="userName" size="10"/>  
密 码: <input type="password" name="password" size="10"/>
```

提交



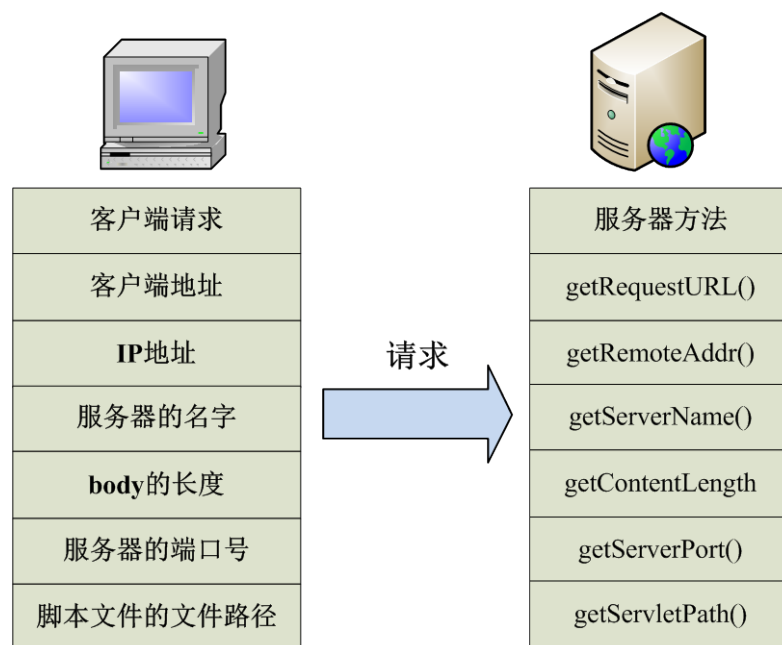
名称	值
UserName	lester
password	123

服务器接收

```
"表单输入userName的值:"+request.getParameter("userName")  
"表单输入password的值:"+request.getParameter("password")
```

3.1.2 获取服务器端和客户端信息

- * 使用request对象回可以获取提交请求的客户端信息及接收请求的服务器端信息，这些获取信息的方法如图3.6所示。



3.1.3 request 中保存和读取共享数据

- * request对象不仅能够封装请求信息，而且还可以保存和读取某一范围内的共享数据。request对象定义了一对方法getAttribute(String name)和setAttribute(String name, java.lang.Object objt)，用来在request对象中读取和保存数据。

3.2 response 内置对象

- * response对象是服务器端向客户端返回的数据，从这个对象中间可以取出一部分与服务器互动的数据和信息。response对象的主要方法及方法描述如表3.2所示。

方法	方法描述
addCookie(Cookie cookie)	添加一个Cookie对象，用来保存客户端信息
addHeader(String name, String value)	添加HTTP文件头信息，如果已有同名的Header，则覆盖它
containsHeader(String name)	判断名字为name的HTTP文件头是否已存在
flushBuffer()	强制将当前缓冲区的内容发送到客户端
getBufferSize()	返回缓冲区的大小
getOutputStream()	获取到客户端的输出流对象
sendError(int)	向客户端发送错误信息
sendRedirect(String location)	发响应发送到另一个位置去处理
setContentType(String contentType)	设置响应的MIME类型
setHeader(String name, String value)	设置名字为name的HTTP文件头的值，新设置的值可以覆盖旧值

3.2.1 response实现页面转向

- * 使用response对象的sendRedirect(String location)方法可以实现页面的转向。在上一章的动作指令中，我们也学过<jsp:forward>动作指令也能够实现页面的转向，那么这两种转向有什么不同呢？它们的区别如图3.9所示。

<jsp:forward>

(1) JSP引擎控制权的转向，地址栏中不会显示转向后地址

(2) 转向地址必须是相对路径，转向页面与转向到页面必须位于一个Web应用中

sendRedirect()

(1) 完全跳转，浏览器将会得到跳转后地址，并重新发送链接

(2) sendRedirect()方法中location用来指定转向地址。既可以是相对路径，也可以是一个合法的URL

3.2.2 动态设置页面返回的MIME类型

- * 在JSP中可以使用page编译指令来设置页面的MIME（Multipurpose Internet Mail Extensions 多功能Internet邮件扩充服务，即文件的类型）返回类型，但是在这里设置是页面的编译阶段，以电脑设置完成，在运行阶段是不可更改的。而使用response对象中的setContentType(String type)方法可以来动态设置页面的返回类型。

3.3 out 内置对象

- * out 内置对象是在Web应用开发过程中使用最多的一个对象，其功能就是动态的向JSP页面输出字符流，从而把动态的内容转化成HTML形式来展示。这个对象在任何JSP页面中都可以任意访问。out对象的方法主要用于输出各种各样格式的数据，如表3.3所示。

方法	方法描述
clear()	清除缓冲区的数据，但是仅仅是清除，并不向用户输出
clearBuffer()	清除缓冲区的数据，同时把这些数据向用户输出
close()	关闭out输出流
flush()	输出缓冲区的内容
isAutoFlush()	判断是否为自动刷新
print(String str)	输出带HTML格式的各种类型的数据，下一个输出语句不换行
println(String str)	输出带HTML格式的各种类型的数据，下一个输出语句换行

- * 在out对象方法中，最常用的就是print()和println()方法。我们可以运用这两种方法实现各种类型数据的输出。

3.4 session内置对象

- * session对象用来保存每个用户的信息。例如，登录名、密码、上次访问时间等，从而可以跟踪每个用户的操作状态。一般情况下，当用户首次登录系统时，Web容器就会给该用户创建一个唯一用来标识该用户会话的session id。为了跟踪用户的操作状态，在多个页面之间保存共享信息，JSP中提供了session对象。当该用户退出系统时，这个session自动消失。session对象的主要方法如表3.4所示。

方法	方法描述
getAttribute(String name)	从session中获取名字为name的属性
getAttributeNames()	返回存储在session对象中的所有属性的名字，结果为一个枚举类型
removeAttribute(String name)	删除名字为name的属性
setAttribute(String name, Java.lang.Object value)	设置一个名字为name的属性，其值为value
getCreationTimes()	返回该session被创建的时间
getId()	返回唯一标识该session的ID
getLastAccessedTime()	返回与该session相关的客户端最后发送请求的时间

- * 下面我们通过几个session常用实例的介绍，来详细说明这些方法的具体用法。

3.4.1 获取session的ID

- * session对象的ID是用来唯一识别session的标识。该ID由一个32位的十六进制字符串组成，可以保证服务器中所创建的所有session对象都不相同。

3.4.2 session中保存和读取共享数据

- * 与request对象一样，session对象也有一对setAttribute()和getAttribute()方法，用来存储或者读取session中的共享信息。而两种对象的两个方法的区别在于共享信息的范围不同，session对象中保存的共享信息的范围是整个会话过程，而request对象中保存共享信息的范围则是提交和被提交的页面。

3.4.3 session对象的生命周期

- * session对象的创建是由服务器完成的，当客户端第一次请求服务器时由服务器创建。如果会话过程一直存在，则session对象也将一直存在下去。只有当session过期、客户端关闭浏览器或者服务器端调用了session的invalidate()方法时session对象才被释放掉，结束其生命周期。

3.5 application内置对象

- * application对象保存着整个Web应用运行期间的全局数据和信息。从Web应用开始运行开始，这个对象就会被创建。在整个Web应用运行期间可以在任何JSP页面中访问这个对象。所以如果要保存在整个Web应用运行期间都可以访问的数据，这时候就要用到application对象。
- * application对象常用的主要方法及方法描述如表3.5所示。

方法	方法描述
getAttribute(String name)	返回application对象中名字为name的属性的值
getAttributeNames()	返回application对象中所有属性的名字，结果为一个枚举类型
getInitParameter(String name)	返回application对象中名字为name的属性的初始值
getServletInfo()	返回Servlet编译器的当前版本的信息
setAttribute(String name,Object object)	在application对象中设置一个名字为name的属性，其值为object

- * application对象最常用的方法回是getAttribute()和setAttribute()方法。

3.6 其他内置对象

- * 前面讲解的五种内置对象是在JSP中最为常用的对象，需要读者熟悉并运用。还有四种JSP内置对象使用几率较小，我们只介绍其基本用法。

3.6.1 pageContext内置对象

- * pageContext对象又被称为JSP作用域通信对象。该对象提供了访问其他内置对象的统一入口，使用户可以方便地访问页面作用域中定义的所有内置对象。pageContext对象的主要方法及方法描述如表3.6所示。

方法	方法描述
getRequest()	返回当前页面的request对象
getResponse()	返回当前页面的response对象
getServletConfig()	返回当前页面的servletConfig对象
getServletContext()	返回当前页面的ServletContext对象，这个对象是所有的页面共享的
getSession()	返回当前页面的session对象
setAttribute()	设置默认页面范围或特定对象范围之中的的对象
removeAttribute()	删除默认页面对象或特定对象范围之中的已命名对象

3.6.2 config内置对象

- * config对象代表当前JSP页面的配置信息。但JSP页面通常无须预先进行配置，也就不存在配置信息了。因此该对象在JSP页面中比较少用，但在Servlet中则用处相对较大，因为Servlet需要在web.xml文件中进行配置，从而设置初始化配置参数。config对象常用的方法及方法描述如表3.7所示。

方法	方法描述
getInitParameter(String name)	返回String类型的初始化参数
getInitParameterNames(String name)	返回所有初始化参数的名称
getServletName()	获得当前JSP页面名称
getServletContext()	获得当前JSP页面的服务器上下文环境

3.6.3 exception 内置对象

- * exception对象用来封装运行时出现的异常信息。该对象只能被处理错误的页面使用，一般用来处理错误的页面会在其页面指令中声明“isErrorPage=true”。exception对象的主要方法和方法描述如表3.8所示。

方法	方法描述
getMessage()	返回描述异常的消息
toString()	返回关于异常的简短描述消息
printStackTrace()	显示异常及其栈中的跟踪信息

3.6.4 page 内置对象

- * page 内置对象指向当前JSP页面本身，有点类似于类中的this指针，它表示当前JSP页面转换后生成的Servlet类的实例。page对象常用的方法及方法描述如表3.9所示。

方法	方法描述
getClass()	返回当前Object的类
toString()	返回当前Object对象的字符串
hashCode()	返回当前Object的哈希代码
equals(Object o)	比较当前对象与给定的对象是否相等
copy(Object o)	把当前对象赋值到给定的对象中去
clone()	对当前对象进行克隆操作

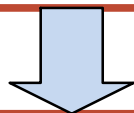
3.7 JSP 中的中文乱码问题

- * 在Java开发中，中文乱码是一个最让人头疼的问题，如果不对中文做特殊的编码处理，这些中文字符就会变成乱码或者是问号。而在不同情况下对这些乱码的处理方法又各不相同，这就导致很多初学者对中文乱码问题束手无策。其实造成这种问题的根本原因是Java中才用的默认编码方式是Unicode，而中文的编码方式一般情况是GB2312,因为编码格式的不同，导致在中文不能正常显示。
- * 本节我们将对JSP开发过程中的中文乱码常见问题进行介绍，并提供对应的解决方法。UTF-8、GBK、GB2312是三种支持中文显示的编码方案，在本节我们统一采用GB2312的编码格式。

3.7.1 JSP页面中文乱码

- * 在JSP页面中，中文显示乱码有两种情况：一种是HTML中的中文乱码，另一中是在JSP中动态输出的中文乱码。
- * 显然，图3.30所示并非我们预期的效果，在JSP源代码中清清楚楚看到的是中文，在这里为什么就成了乱码，造成这种原因的可能就是出在浏览器端的字符显示设置上，我们需要对其进行如图3.31所示的改进。

```
<%@ page language="java" import="java.util.*"%>
```



```
<%@ page language="java" import="java.util.*"  
contentType="text/html;charset=gb2312"%>
```

3.7.2 表单提交中文乱码

- * 对于表单中提交的数据，可以使用`request.getParameter("")`的方法获取。但是当表单中如果出现中文数据的时候就会出现乱码。这是我们在JavaWeb开发中经常会遇到的情况。
- * 我们可以从图3.35中清楚的看到输入的中文用户名在用request取出以后全部变成了乱码，造成这个问题的原因是：在Tomcat中，对于以POST方法提交的表单采用的默认编码为ISO-8859-1，而这种编码格式不支持中文字符。要解决这个问题，我们可以采用转换编码格式的方法，转换方法如图3.36所示。

```
new String(userName.getBytes("ISO-8859-1"), "gb2312")
```

从ISO-8859-1格式的字符串中取出字节内容

然后再用gb2312的编码格式重新构造一个新的字符串

3.7.3 URL传递参数中文乱码

- * 在一般情况下，我们可以采用类似 `http://localhost:8080/JSPWeb/URLCharset.jsp?param='中文'` 这种形式来传递参数。但是这种传递方式仍然有可能会发生乱码问题。
- * 对于URL传递中文参数乱码这个问题，其处理方法比较独特，仅仅转换这个中文字符串的编码，或者设施JSP页面显示编码都是不能解决问题的。在这里需要多Tomcat服务器的配置文件进行修改才可以解决问题。在这里需要修改Tomcat的conf目录下的server.xml配置文件。修改方法是在`port="8080"`后面添加URI编码设置`URIEncoding="gb2312"`即可。如图3.40所示。

```
<Connector port="8080" protocol="HTTP/1.1"  
connectionTimeout="20000" redirectPort="8443" />
```

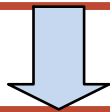


```
<Connector port="8080" URIEncoding="gb2312" protocol="HTTP/1.1"  
connectionTimeout="20000" redirectPort="8443" />
```


3.7.4 MyEclipse开发工具中文JSP文件的保存

- * 在Eclipse中，JSP文件默认的编码格式为ISO-8859-1，所以在JSP代码中间如果出现中文就不能保存。
- * 对与这个问题，只要在JSP页面中指明页面编码即可。pageEncoding="gb2312"指明了JSP页面编码采用gb2312，这样就可以正常保存JSP的源文件。修改方式如图3.43所示。

```
<%@ page language="java" import="java.util.*" %>
```



```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
```

3.8 小结

- * 本章主要讲解JSP内置对象的内容，包括了JSP九大对象，其中重点讲解了前5种类型。由于在JSP编码中读者经常会遇到中文乱码问题，所以我们又增加了一节对于这个问题解决方法的讲解。本章的重点是对request、response、session、out这些重要内置对象的理解，难点是能在理解的基础上实现对这些对象的熟练运用。希望读者多加练习，在今后的Web开发中正确地掌握和灵活地使用JSP内置对象。

第4章 JavaBean基础

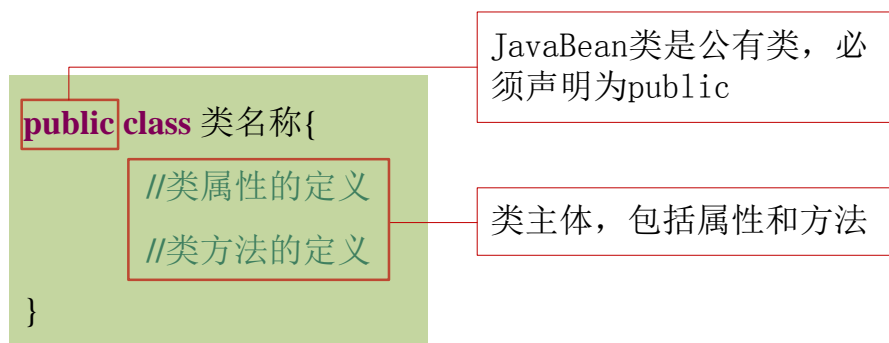
- * JavaBean是一种Java语言写成的可重用组件。JSP可以方便地支持JavaBean组件的使用。用户将常用的功能写入JavaBean。当用户需要使用这些功能，直接在JSP页面调用对应的JavaBean即可。实现了一次编写，任何地方调用。本章将详细讲解如何编写JavaBean，以及JSP如何调用JavaBean，最后我们会为大家展示JavaBean在Web领域的具体应用。

4.1 创建JavaBean

- * Sun公司对JavaBean的定义为：可以重复利用的软件组件，它在遵循JavaBean技术规范的基础上提供特定的功能，这些功能模块可以组合成更大规模的应用系统。JavaBean其实本质上就是一个封装了一系列属性和方法的类。其中属性和方法封装需要遵循各特定的规范。本节将讲解如何创建JavaBean。

4.1.1 JavaBean类

- * 首先我们要创建一个JavaBean类。JavaBean类创建的语法格式如图4.1所示。

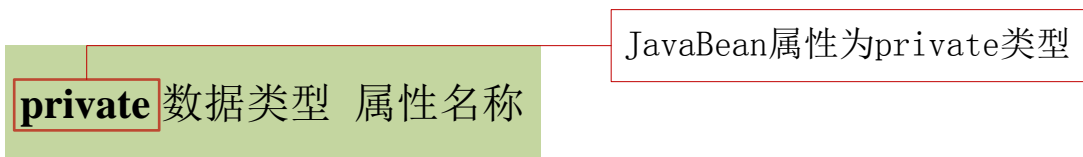


4.1.1 JavaBean类

- * 一个标准的JavaBean类有以下几个特性：
- * 它是一个公开的（public）类。
- * 它有一个默认的构造方法，也就是不带参数的构造方法（在实例化JavaBean对象时，需要调用默认的构造方法）。
- * 他提供getXXX()和setXXX()方法来让外部程序设置和获取JavaBean的属性。
- * 一般来说，符合上述条件的类，我们都可以将其看做JavaBean类。
- * 明白了如何创建JavaBean类后，我们再来看如何创建JavaBean的属性和方法。

4.1.2 JavaBean属性和方法

- * JavaBean的属性用于表示其内部状态。在Java Web开发中，其属性主要用来存储中间数据。JavaBean属性定义如图4.2所示。



- * 对于我们在JavaBean中生命的属性，在类中必须定义用来获取或更改属性值的两个方法——getXXX()和setXXX()方法。
- * JSP文件就运用JavaBean方法在需要时从JavaBean中把这些属性取出，然后在客户端将其显示出来。根据JavaBean类特定的接口格式要求我们可以将其属性分为简单方法和索引方法。

1. 简单方法

- * 简单的方法是指一个拥有get或者set方法的方法。我们在Java Web开发中使用的JavaBean属性一般都是读写类型，必须采用标识命名约定来定义getXXX()和setXXX()方法。对于布尔类型的值我们还可以采用is()属性来获取属性值。简单方法的使用语法如图4.1所示。

```
public void set<PropertyName>(PropertyType value);
```

set方法用来设置属性的值

```
public <PropertyType> get<PropertyName>();
```

get方法用来获取属性的值

```
public Boolean is<PropertyName>();
```

is()方法Boolean类型属性用来获取属性值

2. 索引方法

- * 索引方法是指一个有get/set方法的数组方法。get和set方法的作用同简单类型的方法一样，即用来获取和设置属性值。但是索引方法不只有一个get或者set方法，可能有两个get方法，但是参数不一样。索引方法的语法格式如图4.3所示。

```
public void set<PropertyName>(int index, <PropertyType> value);  
public void set<PropertyName>(<PropertyType[]> value);  
public <PropertyType[]> get<PropertyName>();  
public <PropertyType> get<PropertyName>(int index);
```

用来设置属性的值

用来获取属性的值

4.2 JSP与JavaBean交互的动作指令

- * 在JSP中专门提供了3个动作指令来与JavaBean进行交互，分别为<jsp:useBean>动作指令、<jsp:setProperty>动作指令和<jsp:getProperty>动作指令。

4.2.1 <jsp:useBean>动作指令

- * <jsp:useBean>动作指令用来在JSP页面中获取或创建一个JavaBean组件的实例并指定它的名字和作用范围。<jsp:useBean>动作指令的语法形式如图4.6所示。

```
<jsp:useBean id="name" scope="page|request|session|application" class="className"/>
```

或者：

```
<jsp:useBean id="name" scope="page|request|session|application" class="className">
```

```
body //执行语句
```

```
</jsp:useBean>
```

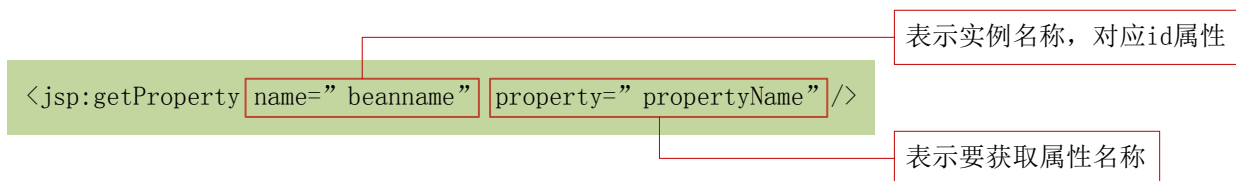
4.2.1 <jsp:useBean>动作指令

- * 该动作指令表示的含义是：在页面中引用一个已经存在或创建一个新的由class属性指定的Java类的实例，然后将其绑定到名字由id属性给出的变量上，并且该变量只在scope属性所指定的范围内有效。对于第二种形式，则是只用当第一次实例化JavaBean时，才执行body（JSP语句）部分，如果是获取现有的JavaBean实例，则不执行body部分。然后我们看一下这几个属性的作用，如表4.1所示。

属性名	属性作用
id属性	在定义范围内确认JavaBean实例变量，也可以用该变量名引用JavaBean实例
class属性	引用的JavaBean的完整类名。JSP2.0规范要求JavaBean必须要有包名
scope属性	JavaBean存在范围及id变量名有效范围。翻飞由小到大依次为：page、request、session和application。默认为page

4.2.2 <jsp:getProperty>动作指令

- * 在JSP页面中我们可以通过<jsp:getProperty>和<jsp:setProperty>动作指令来代替一般的get和set方法。<jsp:getProperty>动作指令用来获取JavaBean中指定的属性值并将其转化为一个字符串，然后将其输出到页面中。即其作用相当于前面提到的getXXX()方法。
- * <jsp:getProperty>动作指令的语法格式如图4.10所示。



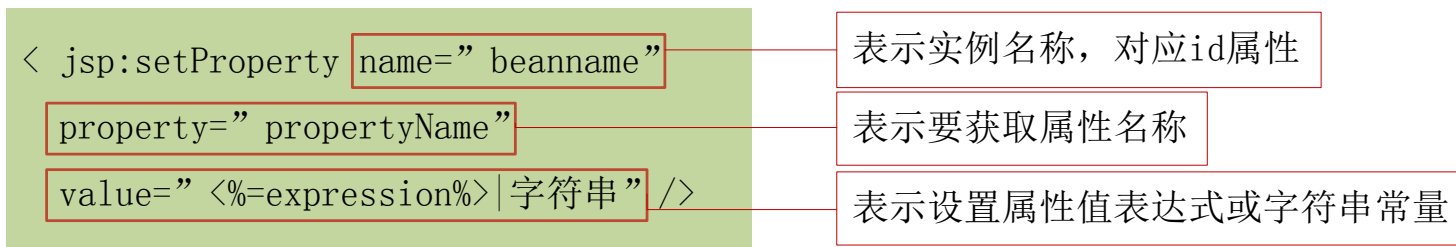
- * 值得一提的是，在使用<jsp:getProperty>动作指令之前，必须使用<jsp:useBean>动作指令来获取或者创建JavaBean实例。

4.2.3 <jsp:setProperty>动作指令

- * <jsp:setProperty>动作指令用来设置已经实例化的JavaBean对象的属性值。实际上，该动作指令作用即相当于获取属性值的setXXX()方法。
<jsp:setProperty>动作指令有3种不同的语法形式。

1. 通过表达式或字符串常量设置属性

* 这种形式的具体语法格式如图4.12所示。



2. 通过内置对象request传递的参数值设置属性

- * 在实际应用中，直接使用表达式或字符串常量设置值的情况很少，往往都是通过接收用户请求中传递的参数值来设置JavaBean属性的。该形式的具体语法如图4.14所示。

```
< jsp:setProperty name=" beanname"  
property=" propertyName"  
param=" paramName" />
```

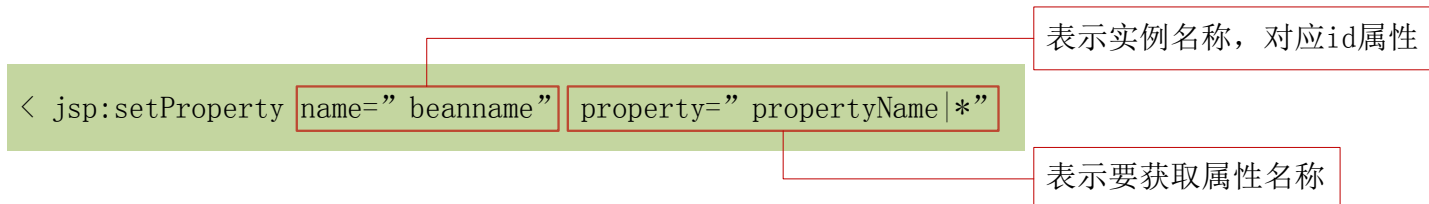
表示实例名称，对应id属性

表示要获取属性名称

表示通过request传递的参数的名字

3. 通过表单的提交参数设置属性

* 这种形式的具体语法形式如图4.18所示。



* 这种形式省略了第二种方式中的param属性。但要求表单中参数名字必须与JavaBean中的名字一致。

4.3 JavaBean的应用

- * 在Web应用中，我们经常要用到JavaBean，其中最常用的有两种——计数器和数据库应用。有关数据库的具体知识我们将在后面单独一章为大家介绍。本节先来介绍这两种功能的简单应用。

4.3.1 计数器JavaBean

- * 对于一个Web应用来说，计数器的功能几乎是必不可少的。接下来我们就为大家介绍如何应用JavaBean实现一个简单的计数器。
- * 这个JavaBean的功能是定义一个计数器变量，并且给出这个变量的取值和赋值的方法。
- * 注意：在这个计数器中，刷新页面不会改变计数器的值，只有新打开一个浏览器窗口这时候才会使计数器的值增加，而且因为这个JavaBean的作用范围是application，所以只要服务器在运行这个计数器的值都会保存在服务器中，当服务器关闭的时候这个值会被置零。

4.3.2 数据库应用

- * JavaBean同样可以使用到数据库开发中，从而简化开发过程，提高代码的可重用性。接下来的将要介绍的内容就是利用JavaBean封装数据库操作。我们首先通过一个例子来说明如何实现JavaBean操作数据库。

4.4 小结

- * 本章主要介绍了JavaBean的属性和方法，并在此基础上介绍了JSP中与JavaBean交互的3个动作指令的具体用法。最后通过实例讲解了JavaBean作为计数器和在数据库中的应用。本章的重点是了解3个动作指令的用法，难点是能够熟练掌握JavaBean在Web中，尤其是数据库中的应用。熟练掌握并运用JSP+JavaBean模式进行Web应用的开发，是目前JSP技术的基本要求，所以读者要多加练习，以打好JSP编程的基础。

第5章 Servlet编程

- * Servlet是Java Web程序的核心。JSP和几乎所有的Java Web框架（如Struts、Webwork）在底层的实现都会看到Servlet的影子。因此，充分了解Servlet的原理和使用方法，对于以后学习Struts等Web框架将起到非常大的帮助。本章我们将为大家介绍Servlet的基础知识，并通过具体的示例介绍Servlet的强大功能。

5.1 Servlet基础

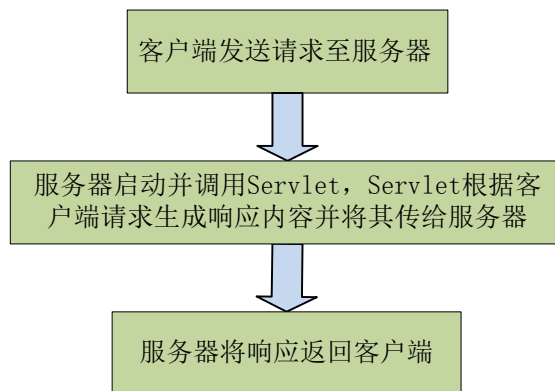
- * Servlet在本质上就是Java类。编写Servlet需要遵循Java的基本语法，但是与一般Java类所不同的是。Servlet是只能运行在服务器端的Java类，而且必需遵循特殊的规范，在运行的过程中有自己的生命周期。

5.1.1 什么是Servlet

- * Servlet是运行于服务器端的、按照其自身规范编写的Java应用程序。我们可以用图5.1来解释这个概念。

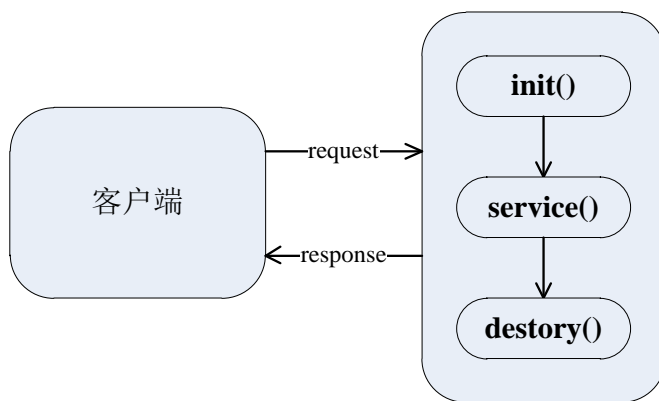
Java类	Servlet是用Java语言编写的，遵守所有Java语言的语法规则的Java类
服务器端运行	Servlet是在服务器端运行的。它编译后的“.class”文件被服务器端调用和执行
必须调用Java Servlet API	Servlet必须调用Java Servlet API，必须是对特定类或接口的继承或实现。并且，它必须重写特定的方法去处理客户端请求

- * Servlet的主要功能是用来接受、处理客户端请求，并把处理结果返回到客户端显示。其作用过程如图5.2所示。



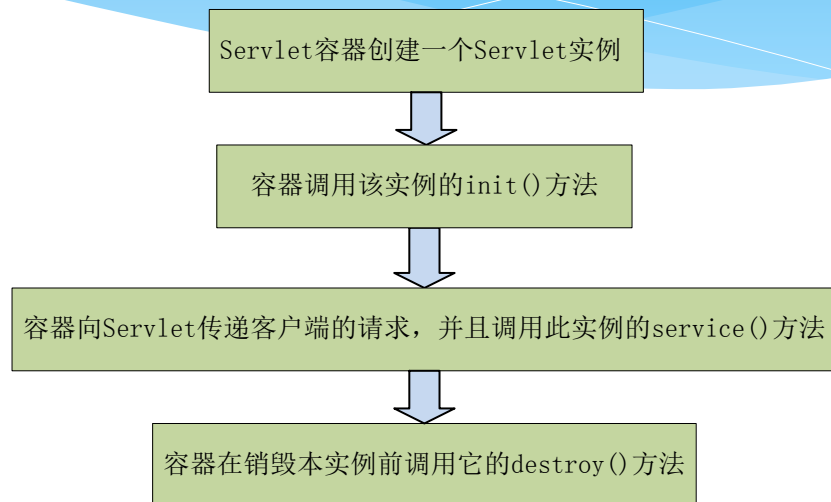
5.1.2 Servlet的生命周期

- * Servlet需要在特定的容器中才能运行，在这里所说的容器即Servlet运行的时候所需的运行环境。一般情况下，市面上常见的Java Web Server都可以支持Servlet，例如Tomcat、Resin、Weblogic、WebSphere等，在本书中采用Tomcat作为Servlet的容器，由Tomcat为Servlet提供基本的运行环境。
- * Servlet的生命周期指的是Servlet从被Web服务器加载到它被销毁的整个生命过程。这个过程如图5.3所示。



5.1.2 Servlet的生命周期

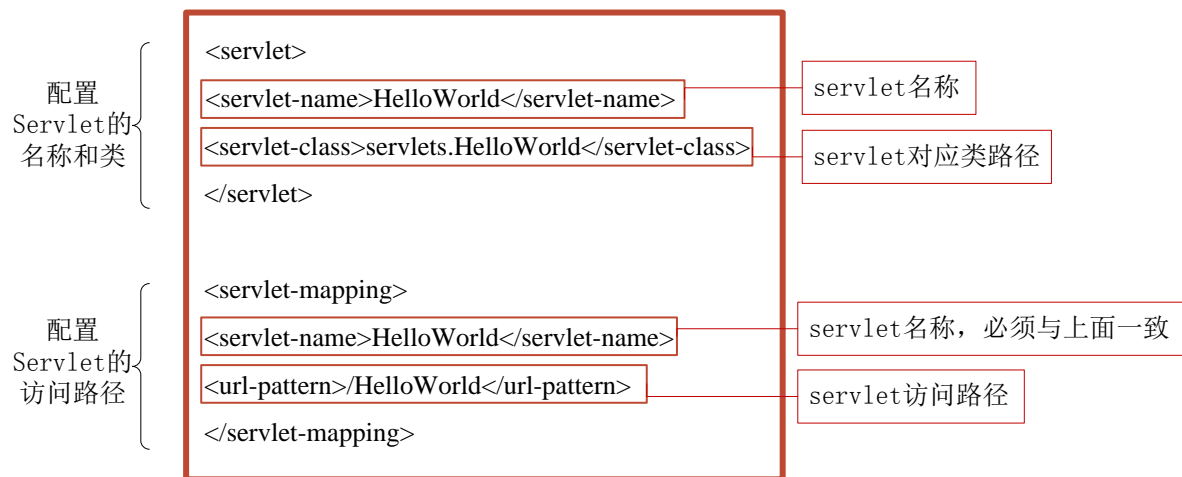
- * 从图5.3中我们呢可以看出，Servlet生命周期的执行大致分为4个步骤，如图5.4所示。



- * 在以上几个阶段中，Servlet对外提供服务阶段是最重要的。
service()方法是编程人员真正要关心的方法。因为它才是Servlet真正开始响应客户端请求，并且处理业务逻辑的方法。service()接收到客户端请求后，再调用该Servlet的相应的方法去处理请求。所以程序员在编写自己的Servlet时，一般只需要重写方法。在该方法中去处理客户端请求，并把处理结果返回。

5.2 简单Servlet开发配置示例

- * 在这一节中我们主要应用Servlet编写一个HelloWorld程序，实现向客户端浏览器中输出“HelloWorld”信息。
- * 在系统中创建的第一个Servlet程序系统会为我们自动生成web.xml配置文件，但是以后的Servlet程序就需要我们自己去配置了。即一般情况下都需要在当前应用项目的web.xml配置文件中对各个Servlet进行配置，其中web.xml文件的位置在当前项目应用的WEB-INF文件夹下。我们就结合图5.8的示例来讲解如何对Servlet进行配置，如图5.9所示。



5.2 简单Servlet开发配置示例

- * 总之，编写一个Servlet要经过以下三个步骤：
- * （1）编写Servlet的功能代码，即实现功能的代码类。
- * （2）把编译成功的Servlet功能代码类文件拷贝到当前应用项目的WEB-INF/classes目录下。
- * （3）在当前应用项目的web.xml文件中对Servlet进行配置，即在web.xml中添加配置信息。
- * 经过这样三个步骤我们就可以通过浏览器访问这个Servlet了。

5.3 使用HttpServlet处理客户端请求

- * HttpServlet是使用HTTP协议的Web服务器的Servlet类，这个类已经被系统定义好。该类的一些方法，如doGet()方法、doPost()方法等，提供了处理客户端请求的接口。在实际编程中，程序员需要继承这个类，并重写上述方法，去编写自己的Servlet。使用重写后的方法，就可以完成对客户端请求进行处理。

5.3.1 处理Get请求doGet

- * doGet()方法是HttpServlet类中用来处理Get请求的方法。用户通过继承HttpServlet，重写doGet()方法，实现对客户端的Get请求进行处理。要调用doGet()方法，必须在客户端的表单里指定请求的类型为Get。doGet()方法的语法格式如图5.10所示。



5.3.2 处理Post请求doPost

- * doPost()是HttpServlet中用于处理Post请求的方法。如果要调用doPost()方法，必须在表单中指定Post请求。doPost()方法与doGet()方法的用法一般来说没什么区别，doGet()方法用于处理http get请求，doPost()方法用于处理http post请求。至于它们的不同，简单的说，get是通过http header来传输数据，有字数限制，而post则是通过http body来传输数据，没有字数的限制。doPost()方法的语法格式如图5.14所示。

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
```

↓
方法类型

↓
方法名

↓
请求和响应

5.3.2 处理Post请求doPost

- * 下面我们来为大家介绍一下javax.servlet.http提供的HTTP Servlet应用编程接口。创建Servlet，需要扩展HttpServlet类，HttpServlet类包含init()、destroy()、service()等方法，其中init()和destroy()方法是继承的。具体的方法及方法描述如表5.1所示。

方法名	方法描述
init()方法	服务器装入Servlet时执行。可以配置服务器，在启动服务器或客户机首次访问Servlet时装入Servlet
service()方法	Servlet的核心。每当一个客户请求一个HttpServlet对象，该对象的service()方法就要被调用，而且传递给这个方法一个"请求"(ServletRequest)对象和一个"响应"(ServletResponse)对象作为参数
destroy()方法	在服务器停止且卸载Servlet时执行该方法。可以将Servlet作为服务器进程的一部分来关闭
GetServletConfig()方法	GetServletConfig()方法返回一个ServletConfig对象，该对象用来返回初始化参数和ServletContext。ServletContext接口提供有关servlet的环境信息
GetServletInfo()方法	GetServletInfo()方法是一个可选的方法，它提供有关servlet的信息，如作者、版本、版权等

- * 当服务器调用servlet的Service()、doGet()和doPost()这三个方法时，均需要“请求”和“响应”对象作为参数。“请求”对象提供有关请求的信息，而“响应”对象提供了一个将响应信息返回给浏览器的一个路径。

5.4 JSP页面调用Servlet

- * 在上面HelloWorld的示例程序中，我们直接在浏览器中输入具体的地址进行访问。在实际的应用中，不可能让用户在浏览器中直接输入Servlet的地址进行访问。一般情况下，可以通过调用Servlet进行访问，在这里介绍通过提交表单和超链接两种方式调用Servlet。

5.4.1 通过表单提交调用Servlet

- * 在通过提交表单调用Servlet的时候，只需要把表单的action指向对应的Servlet即可。

5.4.2 通过超链接调用Servlet

- * 当用户有输入的内容需要提交给服务器时，我们可以用表单来调用Servlet。如果在没有输入的数据内容需要提交的情况下，我们可以直接通过超链接的方式来调用Servlet，并对其传递参数。

5.5 Servlet 文件操作

- * 在JSP的开发过程中，我们常常把相关内容存储为文件。在Servlet中我们可以使用输入输出流实现对文件的读写。同时，使用Servlet还可以很方便的实现文件的下载操作。这一节我们就来学习如何实现Servlet的文件操作。

5.5.1 Servlet读取文件

- * 我们举一个实例FileRead.java来读取一个文本文件content.txt的内容，并且在页面上打印文件的内容。

5.5.2 Servlet 写文件

- * Servlet写文件的处理方法和读取文件的处理方法非常类似，即把文件输入流换成文件输出流。我们也可以来看一个写文件示例。

5.5.3 Servlet 下载文件

- * 利用Servlet可以很方便地实现文件的下载，我们只需要对服务器的响应对象response进行简单的设置即可。

5.6 Servlet的应用

- * Servlet是与HTTP协议紧密结合的，使用Servlet几乎可以处理HTTP协议各个方面的内容，在本节的几个示例程序中，将集中展示Servlet在HTTP方面的具体应用。

5.6.1 获取请求信息头部内容

- * 当用户访问一个页面的时候，会提交一个HTTP请求给服务器的Servlet引擎，在这个请求中包含了HTTP文件的详细属性信息。我们可以应用 `request.getHeaderNames()` 方法来获取请求信息头部内容。

5.6.2 获取请求信息

- * 在上面的Servlet示例中，我们取出了HTTP文件头信息，在Servlet中还可以很方便取出用户发出请求对象自身的信息。这些信息是和用户的请求密切相关的，例如用户提交请求所使用的协议，客户提交表单的方法是POST还是GET等。

5.6.3 获取参数信息

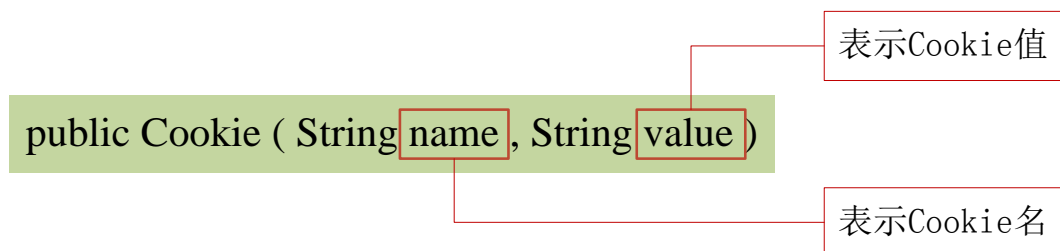
- * 有关用户请求的参数信息，也可以通过Servlet来获取。这种参数既包括以POST方法或者是GET方法提交的表单，也包括直接使用超链接传递的参数。Servlet都可以使用`request.getParameter()`方法取得这些参数信息并且加以处理。

5.6.4 Cookie操作

- * Cookie是一种在客户端保存信息的技术。读者在浏览网页时可能会注意到这样的现象，如在打开某个登录网页时，在第一次打开时，用户名文本框是空的，当输入一个用户名，并成功登录后。在第二次打开这个登录网页时，在第一次输入的用户名会被自动填入这个用户名文本框，就算重启计算机后，仍然如此。其实这就是Cookie所起的作用。

5.6.4 Cookie操作

- * 在Servlet中，使用java.servlet.http.Cookie类来封装一个Cookie消息，在HttpServletResponse接口中定义了一个addCookie方法来向浏览器发送Cookie消息（也就是Cookie对象），在HttpServletRequest接口中定义了一个getCookies方法来读取浏览器发送的Web服务器的所有Cookie消息。Cookie类中定义了生成和提取Cookie消息的各个属性的方法。Cookie类只有一个构造方法，它的语法结构如图5.47所示。



5.6.4 Cookie操作

* Cookie类中的其他常用方法如表5.3所示。

方法名	方法描述
getName方法	用于获得Cookie的名称
setValue和getValue方法	分别用于设置和获得Cookie的值
setMaxAge和getMaxAge方法	分别用于设置和获得Cookie在客户机的有效时间，也就是在在客户机上的有效秒数
setPath和getPath方法	分别用于设置和获得当前Cookie的有效Web路径
setDomain和getDomain方法	分别用于设置和获得当前Cookie的有效域
setComment和getComment方法	分别用于设置和返回当前Cookie的注释部分
setVersion与getVersion方法	分别用于设置和获得当前Cookie的协议版本
setSecure和getSecure方法	分别用于设置和获得当前Cookie是否只能使用安全的协议传输Cookie

5.7 Session技术

- * session对象用来保存每个用户的用户信息和会话状态。session对象由服务器端自动创建，可以跟踪每个用户的操作状态。用户首次登录系统时服务器会自动给用户分配唯一标识的session id，可以用来区分其他用户。相对于Cookie，session是存储在服务器端的会话，相对安全，而且其存储长度限制也大大的扩大了。

5.7.1 HttpSession接口方法

- * 在Servlet中使用HttpSession对象来描述Session。一个HttpSession对象就是一个Session。使用HttpServletRequest接口的getSession方法来获得一个HttpSession对象。
- * HttpSession接口中的主要方法如表5.4所示。

方法名	方法描述
getId方法	用于返回当前HttpSession对象的SessionID
getCreationTime方法	用于返回当前的HttpSession对象的创建时间
getLastAccessedTime方法	用于返回当前HttpSession对象的上一次被访问的时间
setMaxInactiveInterval和 getMaxInactiveInterval方法	分别用来设置和返回当前HttpSession对象的可空闲的最长时间（单位：秒），这个时间也就是当前会话的有效间隔
isNew方法	用来判断当前的HttpSession对象是否是新创建的，如果是则返回true，否则返回false
invalidate方法	用于强制当前的HttpSession对象失效，这样Web服务器可以立即释放该HttpSession对象
getServletContext方法	用于返回当前HttpSession对象所属的Web应用程序的ServletContext对象
setAttribute方法	用于将一个String类型的ID和一个对象相关联，并将其保存在当前的HttpSession对象中
getAttribute方法	用于返回一个和String类型的ID相关联的对象
removeAttribute方法	用于删除与一个String类型的ID相关联的对象

5.7.1 HttpSession接口方法

- * getSession是HttpServletRequest接口的方法，这个方法用于返回与当前请求相关的HttpSession对象，该方法有两种重载形式，它们的定义语法如图5.52所示。

```
public HttpSession getSession();
```

若请求消息中含有SessionID，则返回一个HttpSession对象，如果不包含，就创建一个新的HttpSession对象，并返回

```
public HttpSession getSession(boolean create);
```

若create参数为true，返回一个HttpSession对象。若为false，当不包含SessionID，直接返回null

5.7.2 通过Cookie跟踪Session

- * 客户端必须通过一个SessionID才能找到以前在服务端创建的某一个HttpSession对象。通过SessionID找HttpSession对象的过程也叫做Session跟踪。一般客户端的SessionID通过HTTP请求消息头的Cookie字段发送给服务端，然后服务端通过getSession方法读取Cookie字段的值，以确定是否需要新建一个HttpSession对象，还是获得一个已经存在的HttpSession对象，或是什么都不做，直接返回null。
- * 当HttpSession对象是第一次创建时，向这个对象中写一个字符串值。如果HttpSession对象不是第一次创建，那么就将保存在HttpSession对象中的字符串值输出到客户端。

5.7.3 通过重写URL跟踪Session

- * 如果客户端浏览器不支持Cookie或是将Cookie功能关闭，那么就无法使用Cookie来传递SessionID。为了在这种情况下仍然可以使用Session，Servlet规范提供了一种补充会话管理机制。这种管理机制允许在Cookie无法工作的情况下使用URL参数来传递SessionID。
- * 要想通过URL来发送SessionID，必须要重写URL。HttpServletResponse提供了两个方法用于重写URL，如图5.56所示。

encodeURL方法	用于对所有内嵌在Servlet中的URL进行重写
encodeRedirectURL方法	用于对sendRedirect方法所使用的URL进行重写

5.8 Servlet 过滤器

- * 过滤器是小型的Web组件，它负责拦截请求和响应，以便查看、提取或以某种方式操作正在客户机和服务器之间交换的数据。Servlet过滤器应用非常广泛，有拦截的地方一般都可以用到过滤器。当前Web应用中过滤器已经是不可或缺的一部分。

5.8.1 过滤器的方法和配置

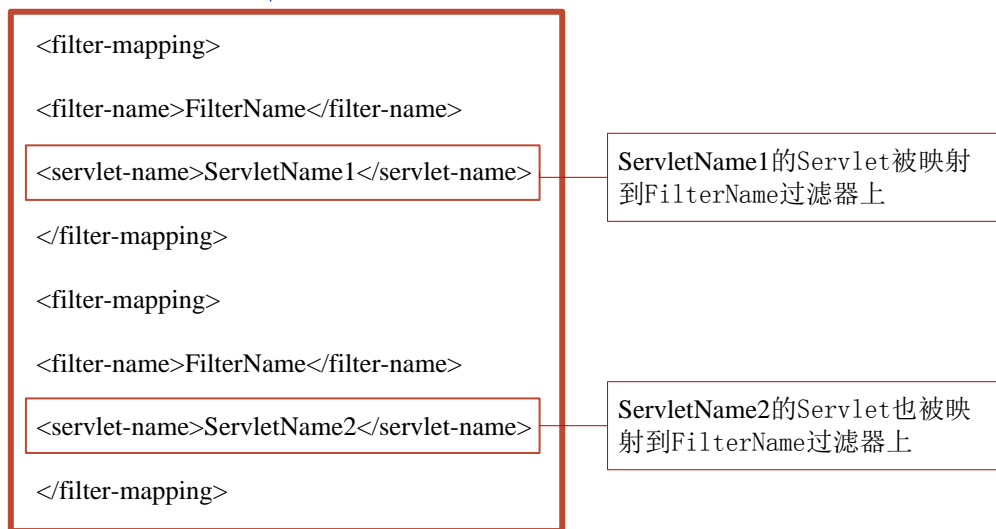
- * 与过滤器相关的Servlet共包含3个简单的接口，分别是Filter、FilterChain及FilterConfig。要实现过滤器功能，必须先实现Filter接口。Filter接口定义了3个方法，如图5.61所示。

init() 方法	在容器实例化过滤器时使用，使过滤器为后面的处理操作做好准备
doFilter() 方法	用于处理请求和响应，当请求与过滤器相关联Web资源时，进行调用
destroy() 方法	Servlet在销毁过滤器实例时调用该方法

- * Servlets过滤器是一个Web应用组件，和Servlet类似，也需要在Web应用配置文件（即web.xml）中进行配置部署。

5.8.1 过滤器的方法和配置

- * 对于过滤器的映射配置，可以将过滤器映射到一个或多个Servlet和JSP文件中。以Servlet为例，我们来看其映射配置，如图5.63所示。



- * 注意：在web.xml中配置Servlet和Servlet过滤器，应该先声明过滤器元素，再声明Servlet元素。

5.8.2 过滤器应用实例——禁止未授权的IP访问站点

- * 在实际的应用中，可能会遇到这样的情况，需要对某些IP进行访问限制，不让非法的IP访问应用系统，这个时候就需要用到过滤器进行限制，当一个用户发出访问请求的时候，首先通过过滤器进行判断，如果用户的IP地址被限制，就禁止访问，只有合法的IP才可以继续访问。

5.8.3 过滤器应用实例——版权过滤器

- * 现在的网页都会在尾部加上版权标志，对于这一操作，我们可以运用过滤器很方便地实现它。

5.9 Servlet监听器

- * Servlet监听器是当今Web应用开发的一个重要组成部分。Servlet监听器主要用来对Web应用进行监听和控制，极大地增强了Web应用的事件处理能力。一般来说，Servlet监听就是指一些特殊的Servlet类，这些类可以监听Web应用的上下文信息、Servlet会话信息、Servlet请求信息。在实际操作中，程序员需要继承或实现一些已定义好的类或接口，从而编写出自己用于监听的类。这些类对特定的信息进行监听。一旦被监听的事件发生，这些类会自动调用相应的方法去执行指定的操作。

5.9.1 监听Servlet上下文信息

- * Servlet上下文信息主要是指关于ServletContext接口的一些信息，比如ServletContext的创建和删除，Servlet属性的增加、删除和修改等。这样就可以实现对Servlet上下文信息的跟踪和记录。为了实现这样的功能，程序员需要实现ServletContextListener和ServletContextAttributeListener接口，从而编写出自己的Servlet类。ServletContext接口的主要方法如表5.4所示。

方法名称	方法描述
getAttribute(String name)	返回Servlet环境对象中指定的属性对象。如果该属性对象不存在，返回空值
getAttributeNames()	返回一个Servlet环境对象中可用的属性名的列表
getContext(String uripath)	返回一个Servlet环境对象，这个对象包括了特定URI路径的Servlets和资源，如果该路径不存在，则返回一个空值
getRealPath(String path)	返回与一个符合该格式的虚拟路径相对应的真实路径的String
getResource(String uripath)	返回一个URL对象，该对象反映位于给定的URL地址的Servlet环境对象已知的资源
getServerInfo()	返回一个String对象，该对象至少包括Servlet引擎的名字和版本号
void log(String msg,Throwable t)	写指定的信息到一个Servlet环境对象的log文件中
setAttribute(String name,Object o)	给予Servlet环境对象中所指定的对象一个名称
removeAttribute(String name)	从指定的Servlet环境对象中删除一个属性

5.9.1 监听Servlet上下文信息

在使用这个监听器之前还需要对Web模块中的web.xml配置文件进行配置，配置代码如图5.75所示。



- * 然后我们就可以编写一个JSP程序testListener.jsp来操作ServletContext的属性，看监听器程序做出什么反应。

5.9.2 监听HTTP会话信息

- * HTTP会话信息指的是Session对象的创建和销毁、会话中属性的设置请求、会话的状态和会话的绑定信息等。通过对HTTP会话信息的监听，可以进行一些很有用的操作，比如，统计当前会话的数目、设置某个对话的属性、了解某个对话的状态等。与ServletContext监听的实现方法类似，对HTTP会话的监听也是通过实现特定的接口来完成的。监听HTTP会话信息需要使用到三个接口类：HttpSessionListener、HttpSessionActivationListener和HttpSessionAttributeListener接口。

5.9.2 监听HTTP会话信息

* 这些接口的主要方法如表5.5所示。

方法名	方法描述
sessionCreated(HttpSessionEvent argo)方法	进行Http会话创建的监听，如果Http会话被创建将调用该方法
sessionDestroyed(HttpSessionEvent argo)方法	对Http会话销毁进行监听，如果某个Http会话被释放将调用该方法
sessionDidActivate(HttpSessionEvent argo)方法	对Http会话处于active情况进行监听
sessionWillPassivate(HttpSessionEvent argo)方法	对Http会话处于passivate情况进行监听
attributeAdded(HttpSessionBindingEvent argo)方法	对Http会话中属性添加进行监听
attributeReplaced(HttpSessionBindingEvent argo)方法	对Http会话中属性修改进行监听
attributeRemoved(HttpSessionBindingEvent argo)方法	对Http会话中属性删除进行监听

5.9.3 对客户端请求进行监听

- * 客户端请求信息是指请求对象的创建、销毁以及其属性的添加、更改和删除。一旦可以对客户端发向服务器的请求进行监听，就可以对它们进行识别，然后统一处理。对客户端请求信息的监听的实现方法与上面两种类似，通过实现ServletRequestListener和ServletRequestAttributeListener接口来完成。这些接口的主要方法如表5.6所示。

方法名	方法描述
ServletRequestListener()方法	监听客户端请求的创建和销毁
attributeAdded(HttpSessionBindingEvent argo)方法	对Http会话中属性添加进行监听
attributeReplaced(HttpSessionBindingEvent argo)方法	对Http会话中属性修改进行监听
attributeRemoved(HttpSessionBindingEvent argo)方法	对Http会话中属性删除进行监听

5.10 小结

- * 本章首先介绍了Servlet编程方面的基础知识，然后在此基础上介绍了Servlet的配置和处理方法，接着我们为大家讲解了如何利用JSP页面调用Servlet和有关Servlet的文件操作，最后我们通过实例讲解了Servlet的具体应用和过滤器、监听器的知识。本章的重点是Servlet的文件操作以及Servlet应用方面的知识，难点是难点是Session技术以及Servlet过滤器、监听器知识的理解和应用。熟练掌握Servlet是学好Java Web技术的基本要求，所以读者要多加练习，以打好基础。

第6章 用户自定义标签

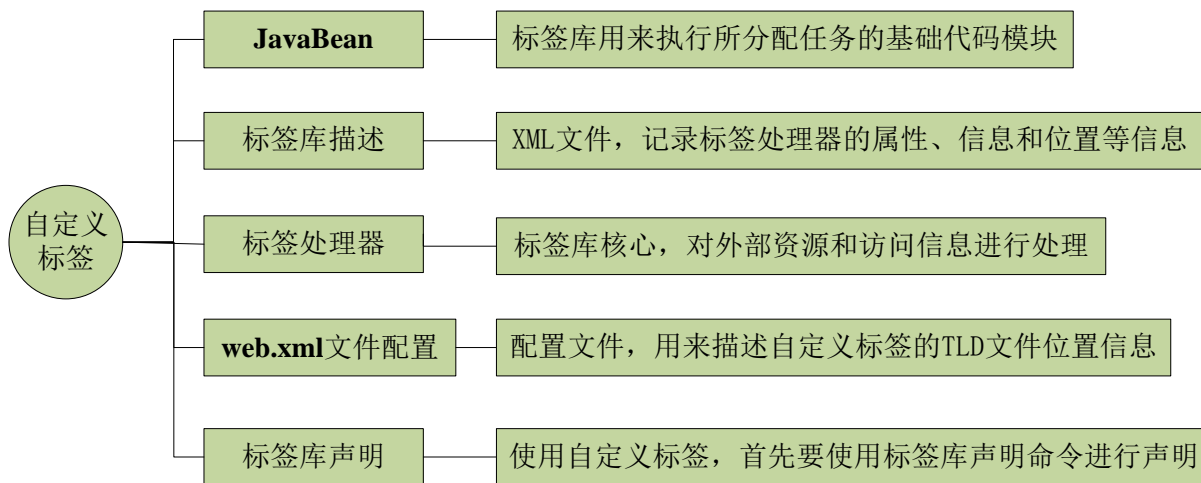
- * JSP自定义标签是用户定义的JSP语言元素，可以看成是一种通过标签处理器生成基于XML脚本的方法。自定义标签在功能上和逻辑上都与JavaBean类似，都是一组可重用的组件代码。相较于JavaBean，自定义标签可以使Web开发者可以完全从Java编程中脱离开来，专注于页面显示和格式上面去，所以具有广阔的发展前景。本章我们就为大家来讲解有关用户自定义标签的知识。

6.1 自定义标签概述

- * 在第4章中我们讲解了JavaBean，知道JSP专门提供了3个动作指令来调用JavaBean组件，简化JSP页面的开发和维护。但是，这远远不能满足实际开发的需要。因此，在JSP技术中提供了一种新的封装动态功能的机制，这就是用户自定义标签。

6.1.1 自定义标签的构成

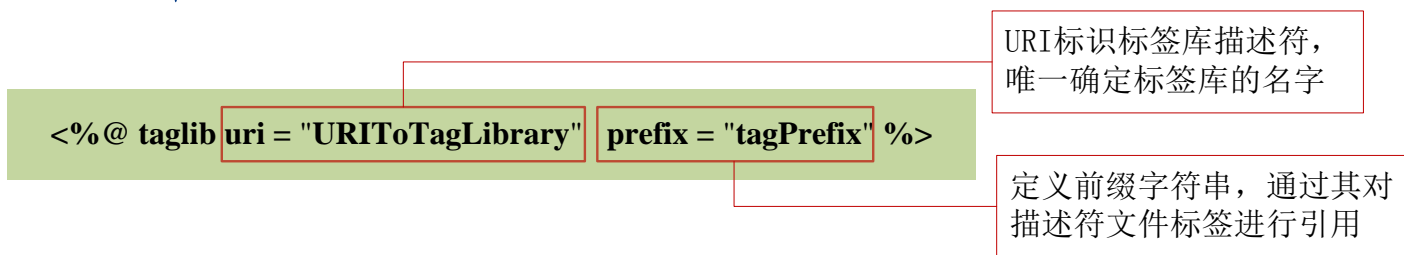
- * 一个自定义标签一般由JavaBean、标签库描述、标签处理器、web.xml文件配置、标签库声明等元素所构成。它们的作用如图6.1所示。



- * JavaBean、web.xml文件配置比较简单。以下仅对标签库声明、标签库描述和标签处理器进行简要介绍。

6.1.2 自定义标签声明

- * 我们在第2章中介绍过taglib指令。该指令就是当JSP页面中引用自定义标签时，用来在页面上对自定义标签进行声明的。taglib编译指令的作用主要是定义一个标签库路径及其前缀。taglib指令的语法格式如图6.2所示。



- * 注意：无论JSP页面中的自定义标签出现在什么位置，taglib指令都必须出现在页面的前端位置。

6.1.3 标签库描述符文件

- * 标签库描述符（TLD文件）是一个描述标签库的XML文档。TLD包含有关整个库以及库中包含的每一个标签的信息。它把自定义标签与对应的处理程序关联起来。TLD文件名称必须扩展名为.tld。TLD文件存储在Web模块的WEB-INF目录下或者子目录下，并且一个标签库要对应一个标签库描述文件，而在一个描述文件中可以包含多个自定义标签的声明。
- * 标签库描述符文件的根元素是<taglib>，该元素下包含如表6.1所示的子元素。

元素	说明	元素	说明
<tlib-version>	用于设置标签库版本	<small-icon>	用于设置标签库的可选小图标
<jsp-version>	用于设置标签库要求的JSP规范版本	<large-icon>	用于设置标签库的可选大图标
<short-name>	用于设置该标签库的助记名	<description>	用于设置标签库的描述信息
<uri>	唯一标识该标签库的URI	<listener>	用于设置标签库的监听器类
<display-name>	用于设置标签库显示的可选名	<tag>	用于设置标签库的具体标签

6.1.3 标签库描述符文件

- * 通过表6.1我们可以看出，<taglib>元素中大部分子元素都是对标签库的一些基本属性或者显示的名称或图表的设定，并不具备实际意义。真正用来查找标签库中具体标签的是<tag>元素。<tag>元素也包括子元素，其具体说明如表6.2所示。

元素	说明	元素	说明
<name>	用于设置标签的唯一名称	<small-icon>	用于设置标签的可选小图标
<tag-class>	用于设置标签处理器的完全限定名	<large-icon>	用于设置标签的可选大图标
<tei-class>	用于设置脚本变量信息的子类名称	<description>	用于设置标签的描述信息
<body-content>	用于设置标签的正文内容类型	<variable>	用于设置标签的脚本变量信息
<display-name>	用于设置标签显示的可选名	<attribute>	用于设置标签的属性信息

6.1.4 标签处理器

- * 把自定义标签的主体和属性转变为HTML代码的实际工作，是由标签处理器来完成的。标签处理器也叫标签处理类，它是一个Java类。当JSP容器编译自定义标签时，就会需要使用标签处理器类的实例。
- * 标签处理器虽然是一个Java类，但不仅仅是一个普通的Java类，在定义时需要满足特殊的要求。开发的标签处理类必须实现Tag或者BodyTag接口类（它们包为`javax.servlet.jsp.tagext`）。BodyTag接口是继承了Tag接口的子接口。如果创建的自定义标签不带体式，可以实现Tag接口，但是如果创建的自定义标签带体，则需要实现BodyTag接口。

6.1.4 标签处理器

* Tag接口类中所定义的方法如表6.3所示。

方法名	方法描述
setPageContext(PageContext pc)	设置当前页面的上下文
setParent(Tag t)	设置这个标签处理类的父类
getParent()	获得父类
doStartTag()	处理这个实例中的开发标签
doEndTag()	处理这个实例中的结束标签
release()	由标签处理类引起，来释放状态

* BodyTag子接口类又重新定义了两个新方法，如表6.4所示。

方法名	方法描述
setBodyContent(BodyContent b)	为体中代码作初始化
doInitBody()	为标签体中的内容设置属性

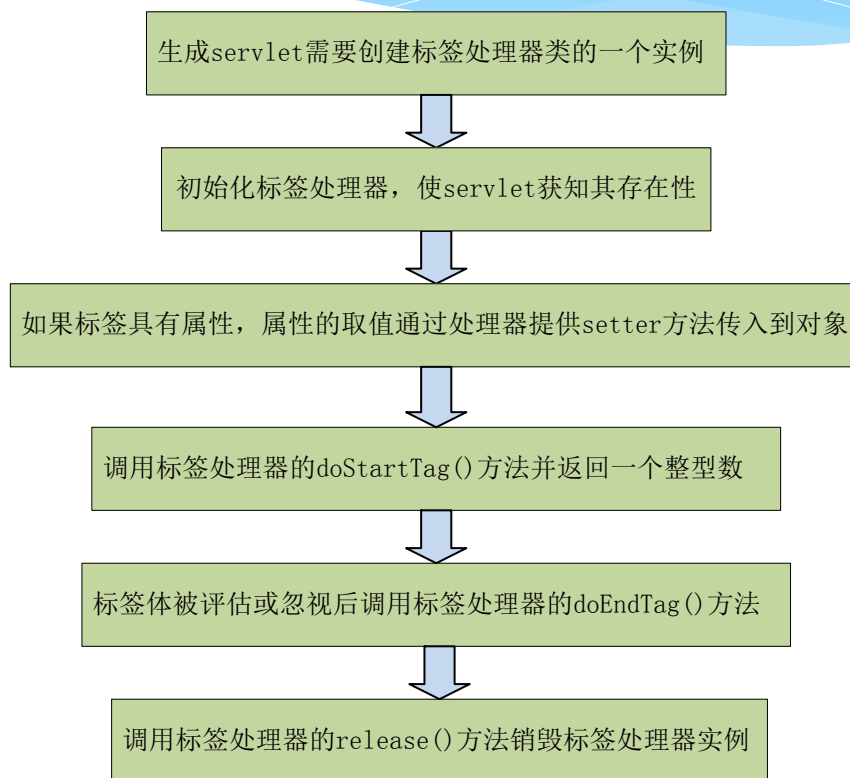
6.1.4 标签处理器

- * 在标签处理器中定义了标签处理方法doStartTag()和doEndTag(), 这两个方法分别在标签开始和结束时执行处理和输出动作。这两个方法都要求分别返回一个状态码, 通知JSP容器对自定义标签的处理结果及整个JSP页面的运行状态。状态码一共有四种, 具体作用如图6.3所示。

EVAL_BODY_INCLUDE	当doStartTag()返回时, 指明servlet应对标签体进行评估
SKIP_BODY	当doStartTag()返回时, 指明servlet应忽视标签体
EVAL_PAGE	当doEndTag()返回时, 指明页面其余部分应被评估
SKIP_PAGE	当doEndTag()返回时, 指明页面其余部分就被跳过

6.1.4 标签处理器

- * 标签处理器也有其生命周期，其大致可以分为5个阶段，如图6.4所示。



6.1.4 标签处理器

- * 下面我们就在自定义标签的基本概念描述的基础上，列举一系列的自定义标签开发的实例，来教会读者如何开发各类自定义标签。

6.2 简单格式的标签开发

- * 简单格式的标签没有属性和体，它必须实现Tag接口中的doStartTag()和doEndTag()方法。当Web容器遇到开始标签时会自动调用doStartTag()方法。由于简单格式的标签没有体，所以这个方法会直接返回一个SKIP_BODY。在遇到结束标签的时候会调用doEndTag()方法。如果还需要页面中的其他部分进行判断，则doEndTag()方法会返回EVAL_PAGE，否则，会返回SKIP_PAGE。

6.3 自定义带有属性的标签

- * 自定义标签可以有自己的属性。属性一般在开始标记中定义，语法为`attr="value"`。而且对于每个value属性，还需要在这个标签相对应的处理类中定义一个属性的`set()`和`get()`方法。

6.4 自定义带有体的标签

- * 之前我们定义的标签都是不带体的，接下来我们看如何创建一个带体的标签。一个自定义标签可以包含其他自定义标签、脚本变量、HTML标记或其他内容。而且其必须继承 `javax.servlet.jsp.tagext.BodyTagSupport` 类，实现其中的 `doInitBody()` 和 `doAfterBody()` 方法。这两种方法的说明如表6.5所示。

方法名	方法描述
<code>doInitBody()</code> 方法	用这个方法执行所有依赖于正文内容的初始化，对正文内容进行判断之前调用
<code>doAfterBody()</code> 方法	返回指明是否继续判断体中正文内容的指示，在判断了正文内容之后调用

6.5 自定义嵌套标签

- * 到目前为止，我们创建的标签都是单个的标签，也是被单独的应用在JSP页面中。而在实际开发中，往往需要通过多个标签来实现特定的功能，这样的标签就存在嵌套关系。存在嵌套关系的标签也可以被称为父子标签，一个父标签可以嵌套多个子标签、HTML和Java片段代码。

6.6 小结

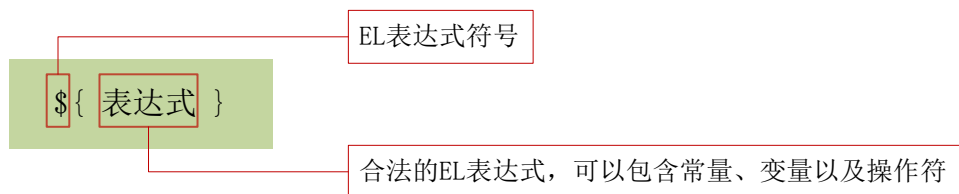
- * 本章主要讲解了JSP自定义标签的基本开发流程，并且通过具体实例详细介绍了各种类型自定义标签在具体实现时的技术细节和应用场合。本章的重点和难点都是熟练掌握包括带有属性、带有体以及含有嵌套的标签的定义和使用方法。通过本章的学习，读者应对JSP技术有了更深层次的理解，并为后面学习JSTL标签库及各种Java Web框架中的标签库打下扎实的基础。

第7章 EL与JSTL

- * JSTL (JSP Standard Tag Library, JSP标准标签库)是一个不断完善的开放源代码的JSP标签库,是由apache的jakarta小组来维护的。JSTL标签库的出现,不仅简化了JSP和WEB应用程序的开发,而且在应用程序服务器之间提供了一致的接口,最大程度地提高了WEB应用在各应用服务器之间的移植。而由JSTL 1.0发展而来的EL语言更加简化了JSP页面中对变量和对象的访问操作。本章我们就来带领大家一起学习EL表达式语言和JSTL标签库的相关知识。

7.1 EL简介

- * 表达式语言是JSP 2.0的一个新特性，全名为Expression Language，简称EL。EL能实现对pageContext对象、session对象、request对象等存储对象的简化访问，能够简洁地访问请求参数、Cookie和其他请求数据，即EL可以很方便地访问大多数JSP内置的隐含对象，从而简化编程。此外，EL还可以简化对JavaBean属性和集合元素的访问。
- * EL语法很简单，使用非常方便。其语法格式如图7.1所示。



- * 我们先来看几个使用EL表达式的示例：
- * `${ 1+2+3 }` //计算1+2+3的值并将结果返回
- * `${ username }` //查找并返回username的值
- * `${ user.name }` //访问JavaBean对象user的属性name

7.1 EL简介

- * EL语言中操作对象时，可以非常简单地使用各种算术、关系、逻辑或空值测试运算符，简化操作运算。如果要针对不同情况和条件进行输出不同的信息，根本不再需要采用Java语言编程，就可以轻松实现条件化输出，从而消除了大部分类型转换的需要，同时也省略掉很多将字符串解析成数字的代码，实现了自动类型转换。例如，实现将用户输出的参数加5后的和输出到页面上这一功能，使用JSP传统语法的具体代码如图7.2所示。

```
<%  
    String str_count = request.getParameter("count");  
    int count = Integer.parseInt(str_count);  
    count = count+5;  
    out.println("count:"+count);  
%>
```

- * 而使用EL实现同样的功能只需要如下简单的一行代码：
- * count: \${ param.count + 5 }

7.1 EL简介

- * 大多数Java Web服务器都是默认支持EL的。对于单个JSP页面，可以使用page指令来设置JSP页面是否支持EL。JSP页面默认支持EL，如果不支持的话，我们可以通过设置page指令的isELIgnored属性为false，来实现对EL的支持。其具体格式如图7.3所示。

```
<%@ page isELIgnored="true/false" %>
```

选择单个JSP页面是否支持EL

- * 注意：isELIgnored属性表示是否忽略EL。由于我们需要使用EL，所以我们将其设置为false。

7.1 EL简介

- * 而对于整个JSP应用，要修改Web应用的web.xml配置文件来设置是否支持EL。如果要使整个JSP应用都支持EL，则设置<jsp-property-group>元素的子元素<el-ignored>的值为false。具体格式如图7.4所示。

```
<jsp-property-group>
```

```
<el-ignored>false|true</el-ignored>
```

选择整个JSP应用是否支持EL

```
</jsp-property-group>
```

- * 注意：el-ignored属性表示整个JSP应用是否支持EL。由于我们需要使用EL，所以我们也将其设置为false。

7.2 EL应用

- * 在JSP中使用表达式语言，可以大大简化JSP开发的工作量。下面我们就从EL在运算符求值、访问作用域变量以及EL内置对象和函数几方面的应用来看EL是如何实现代码简化的。

7.2.1 EL运算符求值

- * EL中的运算符包括算术运算符（+、-、*、/）、关系运算符（>、<）和逻辑运算符（&&、||、!），还有empty运算符用来判断值是否为空。EL中的运算符如表7.1所示。

运算符	代表运算	运算符	代表运算
+	加(算术)	>、gt	大于(比较)
-	减(算术)	<、lt	小于(比较)
*	乘(算术)	<=、le	小于等于(比较)
/、div	除(算术)	>=、ge	大于等于(比较)
%、mod	取模(算术)	==、=	等于(比较)
&&、and	与(逻辑)	!=、ne	不等于(比较)
、or	或(逻辑)	x?y:z	条件求值
!、not	非(逻辑)	empty	检查是否为空

7.2.2 访问作用域变量

- * 可以用EL表达式语言按照pageContext、HttpServletRequest、HttpSession和ServletContext的顺序访问作用域。其一般格式如图7.7所示。

`$\${attrname}$`

作用域变量

- * 它的作用相当于如下代码：
- * `$\${attrname}$`
- * `<%=pageContext.findAttribute(attrname)%>`
- * `<jsp:useBean id="attrname" type="Package.Class" scope=="...">`
- * `<%=attrname%>`

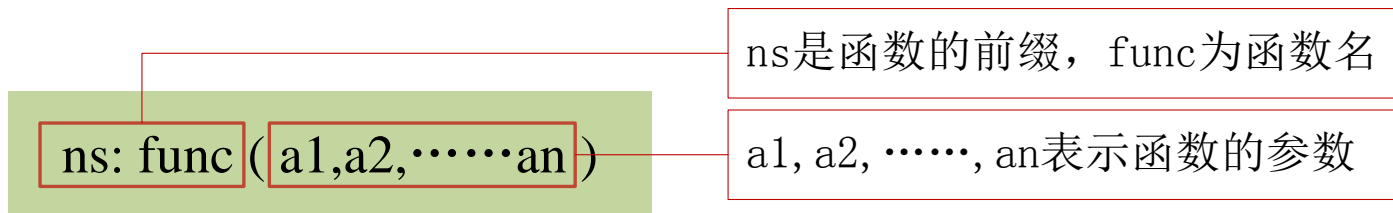
7.2.3 EL内置对象

- * 我们在第2章曾经介绍过JSP的9个内置对象，在EL中也有自己的内置对象。它们共有11个，按功能可以大致分为3类。这11个内置对象的名称和具体说明如表7.2所示。

第一类：pageContext对象。 可以用来访问JSP其他8个内 置对象	pageContext	javax.servlet.ServletContext	可以用于访问JSP的隐含对象
第二类：用于访问环境信息 的对象	cookie	java.util.Map	映射cookie名到单个cookie对象
	initParam	java.util.Map	映射上下文初始化参数名称到单个 值
	header	java.util.Map	映射请求头名称到单个字符串数值
	param	java.util.Map	映射请求参数名到单个字符串参数 值
	headerValues	java.util.Map	映射请求头名称到字符串数组
	paramValues	java.util.Map	映射请求参数名到字符串数组
第三类：用于访问作用域范 围的内置对象	applicationScope	java.util.Map	映射应用程序范围的变量名到其值
	sessionScope	java.util.Map	映射会话范围的变量名到其值
	requestScope	java.util.Map	映射请求范围的变量名到其值
	pageScope	java.util.Map	映射页面范围的变量名到其值

7.2.4 EL函数

- * 表达式语言允许用户自定义函数。此函数必须采用public类中的public static方法编写并要映射到TLD标签库文件中。EL函数的定义语法格式如图7.14所示。



7.3 JSTL简介

- * JSTL的全称是JavaServer Pages Standard Tag Library。它是一个JSP标签的集合，用于简化JSP程序的开发，并使JSP程序更易于维护和管理。JSTL的具体目标是简化JSP页面的设计。对于页面设计人员来说，实用程序语言来操作动态数据是比较困难的，而采用标签和表达式语言则相对容易一些。可以说，JSTL的使用为页面设计人员和程序开发人员的分工协作提供了便利。
- * JSTL实际上是由5个不同的标签库组成的，这样也减少了不同类动作之间的名字冲突。在JSP规范中，围着5个标签库默认指定了默认URI和推荐前缀，如表7.3所示。

标签名	URI	前缀	范例
Core(核心)	http://java.sun.com/jsp/jstl/core	c	<c:tagname>
XML	http://java.sun.com/jsp/jstl/xml	x	<x:tagname>
Internationalization(国际化)	http://java.sun.com/jsp/jstl/fmt	fmt	<fmt:tagname>
Functions(函数)	http://java.sun.com/jsp/jstl/functions	fn	<fn:tagname>
SQL(关系型数据库)	http://java.sun.com/jsp/jstl/sql	sql	<sql:tagname>

7.3 JSTL简介

- * JSTL主要是由Apache组织的Jakarta小组负责实现的，读者可以从Apache网站上下载JSTL安装包，下载地址为<http://tomcat.apache.org/taglibs/standard/>，文件名为jakarta-taglibs-standard-1.1.2.zip。
- * 在lib目录下有两个JAR包：jstl.jar和standard.jar，这两个文件即为所需要的JSTL标签库。将上述两个文件复制到网站根目录下的/WEB-INF/lib/目录下，例如本书中该目录的路径为：
C:\Users\Administrator\Workspaces\MyEclipse
10\JSTL\WebRoot\WEB-INF\lib。重新启动Tomcat，测试JSTL配置是否成功。
- * 在解压出的文件夹中，由一个standard-examples.war文件，这是Jakarta所提供的JSTL的范例程序。我们将其复制到Tomcat的webapps目录下，然后重新启动Tomcat，在浏览器中输入网址<http://localhost:8080/standard-examples/>。
- * 下面我们就来详细介绍JSTL标签库。

7.4 Core标签库（核心标签库）

- * JSTL核心标签库中包含很多标签，根据其功能大致可以分为四类，如图7.23所示。



- * 如果要在JSP页面中使用核心库的标签，需要用taglib指令指明这个标签库的路径为如图7.24所示的形式。

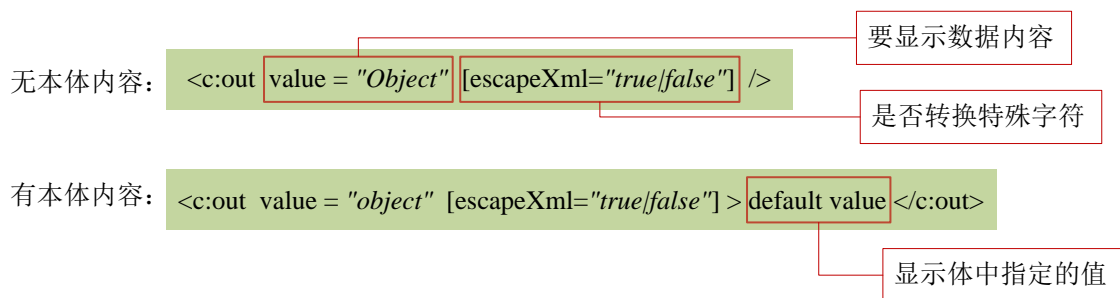


7.4.1 表达式操作标签

- * 表达式操作标签中包含4个标签，分别是：<c:out>、<c:set>、<c:remove>和<c:catch>。

1. <c:out>标签

- * 核心标签库中最为基本的标签就是<c:out>。它可以在页面中显示一个字符串或者一个EL表达式的值。它的功能与JSP传统的<%=表达式%>相类似。<c:out>标签的使用格式如图7.25所示。

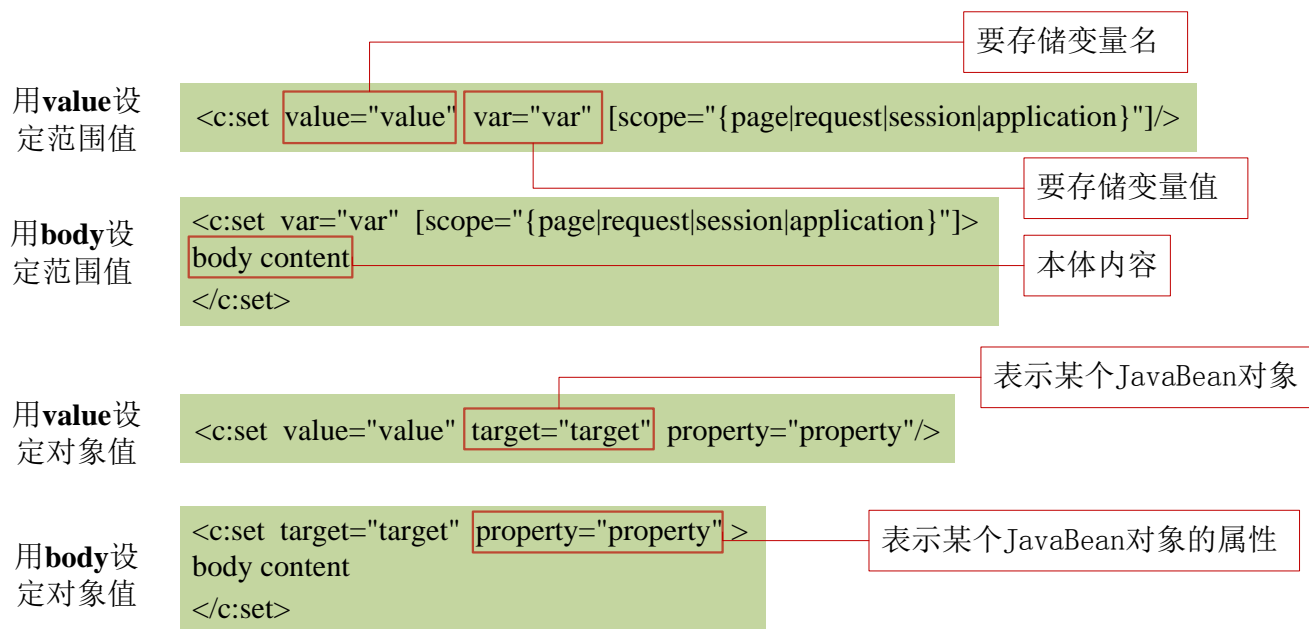


- * <c:out>标签的各个属性的具体描述如表7.4所示。

名称	类型	属性描述
value	Object	输出信息，可为常量值或表达式
escapeXml	Boolean	设置<> & “ ” 这些特殊字符在输出字符串中是否要转换成对应的代码，默认取值为true
default	Object	当value属性为空(null)时，要显示的值

2.<c:set>标签

- * <c:set>标签是用来在某个范围（request、session或者application）内设值，或者设置某个对象的属性值的标签。<c:set>标签的使用格式如图7.28所示。



2.<c:set>标签

* <c:set>标签的各个属性的具体描述如表7.5所示。

名称	类型	属性描述
value	Object	需设定的范围变量或对象属性的值
var	String	要设定的范围变量名
scope	String	var指定变量的范围，默认值为page
target	Object	需设定属性的对象
property	String	需设定值的target对象的属性名称

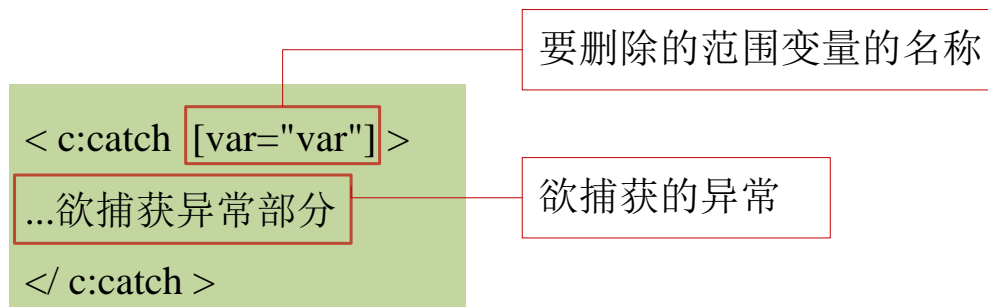
3.<c:remove>标签

- * <c:remove>标签一般和<c:set>标签配套使用，两者是相对应的。<c:remove>标签用于删除某个变量或者属性。<c:remove>标签的使用格式如图7.31所示。



4. <c:catch>标签

- * <c:catch>标签的功能和Java程序中try{}catch{}语句功能很类似，它用于捕获嵌入到它中间语句抛出的异常。<c:catch>标签的使用格式如图7.34所示。

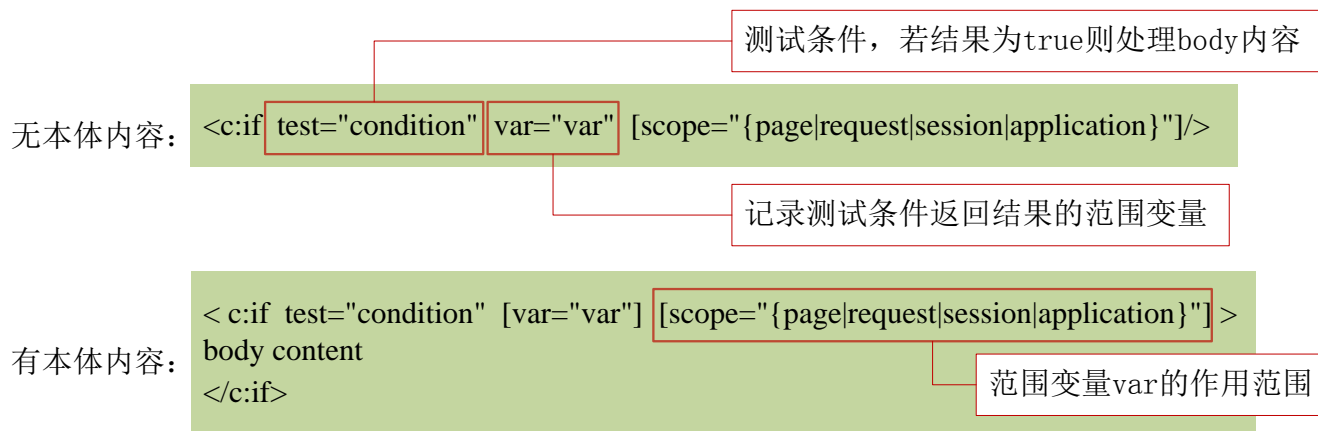


7.4.2 流程控制标签

- * 流程控制分类标签中也包含4个标签，分别是：
<c:if>、<c:choose>、<c:when>和<c:otherwise>。

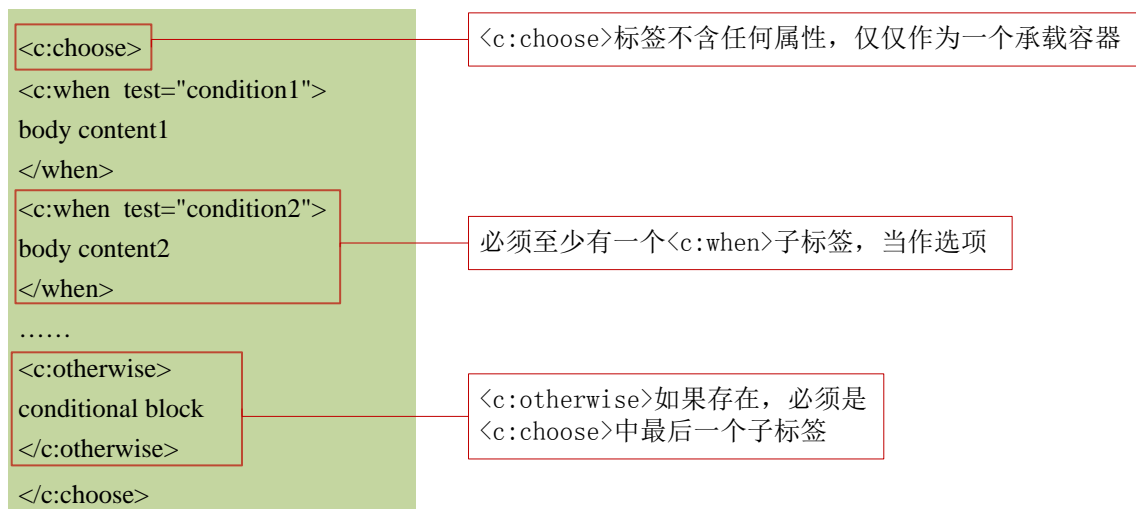
1. <c:if>标签

- * <c:if>标签的作用和Java程序中的if语句作用相同，用于判断条件语句。<c:if>标签使用的格式如图7.38所示。



2. <c:choose>、<c:when>和<c:otherwise>标签

- * <c:choose>、<c:when>和<c:otherwise>标签一般是组合起来一起使用的，就相当于Java程序中的switch条件语句。在<c:choose>标签体中包括<c:when>和<c:otherwise>子标签。<c:when>子标签代表<c:choose>的一个条件分支。这三个标签的使用格式如图7.41所示。



7.4.3 迭代操作标签

- * 迭代分类标签中包含2个标签，分别是<c:forEach>标签和<c:forEachTokens>标签。使用迭代标签可以遍历集合、数组和字符串等，还可以用来指定语句的执行次数。

1.<c:forEach>标签

- * <c:forEach>标签允许用户对一个对象集合执行相同的操作，相对于Iterator迭代器而言，JSTL在很大程度上简化了迭代操作的使用。<c:forEach>标签与Java语言中的for循环有异曲同工之妙，可以用于枚举一个集合对象中的元素，或者用来循环指定的次数。<c:forEach>标签使用格式如图7.44所示。

遍历集合对象

```
<c:forEach [var="varName"] items="collection"  
    [varStatus="varStatusName"] [begin="begin"] [end="end"] [step="step"]>  
    循环体中要执行的内容  
</c:forEach>
```

执行特定次数
的迭代

```
<c:forEach [var="varName"] items="collection"  
    [varStatus="varStatusName"] begin="begin" end="end" [step="step"]>  
    循环体中要执行的内容  
</c:forEach>
```

1.<c:forEach>标签

- * 图中<c:forEach>标签属性的相关描述信息如表7.6所示。

名称	类型	属性描述
var	String	表示当前处理对象的变量
items	支持多种类型，主要包括：Array、Collection、Iterator、Enumeration、Map、string类型	进行循环的项目
varStatus	String	记录循环状态的变量
begin	int	循环的起始点
end	int	循环的结束位置，end的取值不能小于begin
step	int	循环步长

2.<c:forTokens>标签

- * <c:forTokens>标签是JSTL种的另一个迭代循环标签，它可以用来对一个字符串进行迭代循环，这个字符串是用符号分开的。<c:forTokens>标签的使用语法如图7.47所示。

```
<c:forTokens items="stringtokens" delims="delims" [var="var"]  
    [varStatus="varstatus"] [begin="begin"] [end=" end"] [step="step"]>
```

body content

```
</c:forTokens>
```

分隔字符集合

2.<c:forTokens>标签

* <c:forTokens>标签属性的相关描述信息如表7.7所示。

名称	类型	属性描述
items	String	要处理的一系列以特定符号隔开的字符串
delims	String	分隔字符集合
var	String	记录当前处理项目的变量
varStatus	String	记录循环状态的变量
begin	int	循环的起始点
end	int	循环的结束位置
step	int	循环步长

7.4.4 URL相关的标签

- * 核心标签库中还有一些是用来进行与URL相关的操作的，主要包括4个标签，分别是<c:import>、<c:redirect>、<c:url>和<c:param>标签。

1. <c:import> 标签

- * <c:import> 标签用来引入某个URL指向的网页内容到当前JSP网页中。它和传统的JSP标记<jsp:include>相类似，但是有所不同：<jsp:include>标签只能使用来包括该应用中的其他文件，而<c:import>则还可以包含外部站点中的静态文件，所以它的功能更加的强大。<c:import>标签的使用格式如图7.47所示。

网页内容以String
对象的形式输出

```
<c:import url="url" [context="context"][var="var"]  
    [scope="{page|request|session|application}"][charEncoding="charEncoding"]>  
...  
</c:import>
```

网页内容以Reader
对象的形式输出

```
<c:import url="url" [context="context"] varReader="varreader"  
    [charEncoding="charencoding"]>  
...  
</c:import>
```

1. <c:import>标签

* <c:import>标签属性的相关描述信息如表7.8所示。

名称	类型	属性描述
url	String	需要导入的网页URL
context	String	url指定为其他网站的某个URL时，本地网站的路径
var	String	存储导入文件内容的变量
scope	String	变量var的作用域
charEncoding	String	导入文件的字符集
varReader	String	用来读取导入文件内容的Reader对象

2. <c:redirect>和<c:param>标签

- * <c:redirect>标签可以把用户的请求从一个页面转向另一个页面，同JSP中response内置对象的跳转功能类似。<c:param>标签是和<c:redirect>标签一起使用的，它用来进行参数值的传递。<c:redirect>标签的使用格式如图7.51所示。

无参数传递

```
<c:redirect url="url" [context="context"]/>
```

需要重定向到的网页URL位置

有参数传递

```
<c:redirect url="url" [context="context"]>  
<c:param name="name" value="value"/>  
</c:redirect>
```

url指定为其他网站的某个URL时，本地网站的路径

3. <c:url>标签

- * <c:url>标签主要是用来重写URL地址。<c:url>标签也有两种使用语法，如图7.55所示。

无参数传递

```
<c:url value="value" [context="context"] [var="varName"]  
      [scope="{page|request|session|application}"] />
```

存储构造后URL的范围变量

需要构造的URL

有参数传递

```
<c:redirect url="url" [context="context"]>  
<c:param name="name" value="value"/>  
</c:redirect>
```

7.5 XML操作标签库

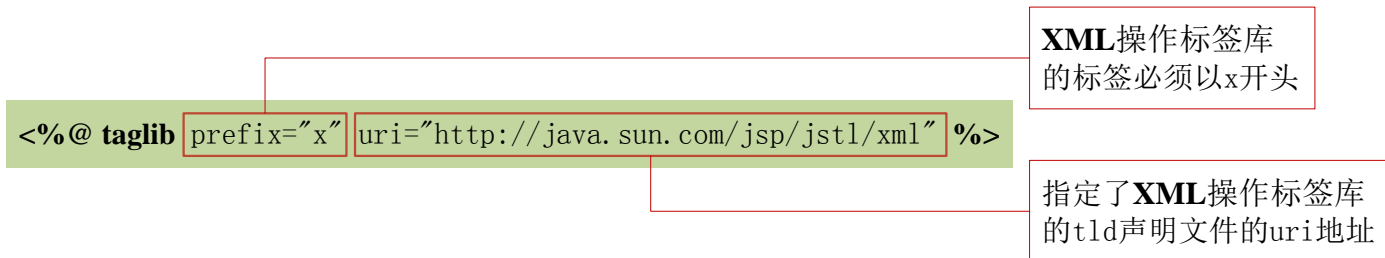
- * 在Java中可以使用SAX或者DOM等API接口来操作XML文档，尽管这种操作功能非常强大而且灵活，但是要很快熟练掌握是有相当大难度的，在JSTL中提供了一组专门处理XML文档的标签，这些标签所提供的功能尽管非常有限，但是已经可以满足基本的XML文档处理需要，而且这些标签学习起来明显比掌握复杂的API接口要容易。接下来的章节中将介绍JSTL中用来处理XML文档的标签。

7.5 XML操作标签库

- * JSTL XML操作标签库中的标签，根据其功能大致可以分为三类，如图7.58所示。



- * 如果要在JSP页面中使用XML操作标签库的标签，需要用taglib指令指明这个标签库的路径为如图7.59所示的形式。



7.5.1 核心操作标签

- * XML核心操作分类标签包含<x:out>、<x:set>和<x:parse>标签。其功能与JSTL中核心标签库中对应的功能基本相同。

1. <x:parse>和<x:out>标签

- * <x:parse>标签可以用来解析一个XML文档。
<x:parse>标签有两种使用语法，如图7.60所示。

以String或Reader
对象指定XML

```
<x:parse {doc="xmlDocument"|xml="xmlDocument"}  
  {var="var" [scope="scope"]|varDom="var" [scopeDom="scopedom"]}  
  [systemId="systemid"]  
  [filter="filter"]/>
```

以body内容指定XML

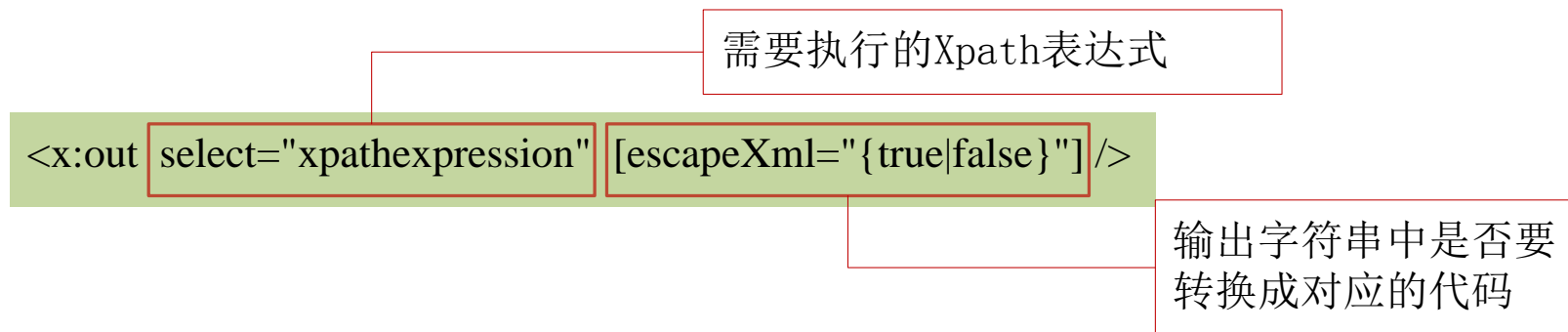
```
<x:parse {var="var" [scope="scope"]|varDom="var" [scopeDom="scope"]}  
  [systemId="systemid"]|filter="filter">  
XML document to parse  
</x:parse>
```

- * <x:parse>标签的属性描述如表7.9所示。

名称	类型	属性描述
doc	String or Reader	需要解析的XML文件
xml	String or Reader	同doc属性，现已不再使用
var	String	用于存储解析后XML文件的范围变量名
scope	String	范围变量var的作用域
varDom	String	用于存储解析后XML文件的范围变量名
scopeDom	String	范围变量varDom的作用域
systemId	String	用于解析XML文件的路径
filter	org.xml.sax.XMLFilter	解析XML文件时使用在XML文件的过滤器

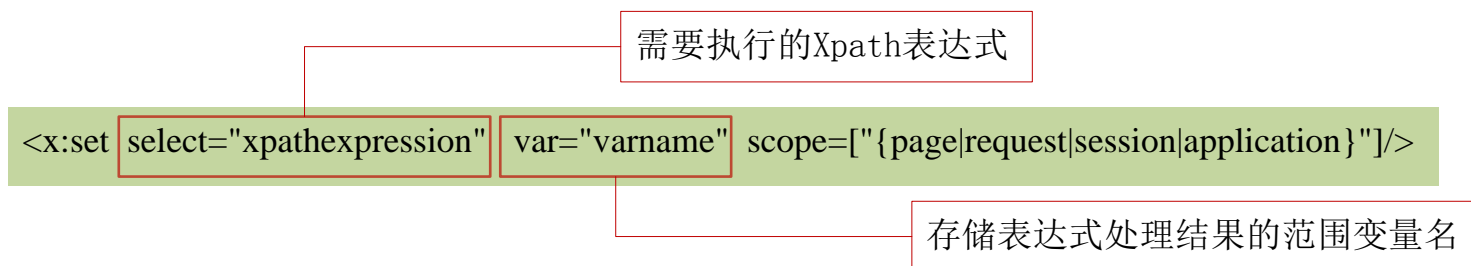
1. <x:parse>和<x:out>标签

- * 使用<x:parse>把一个XML文档解析以后，就可以使用<x:out>标签输出XML节点的值。<x:out>标签用于计算XPath表达式，然后把查找到的元素进行输出。XPath标准是用来对XML文档各部分数据进行定位的语言。<x:out>标签的使用格式如图7.61所示。



2. <x:set>标签

- * <x:set>标签用来把XPath表达式处理的结果存储到某个范围变量中。与<c:set>标签不同之处在于<x:set>标签是把XML中某个节点的内容赋值到一个变量中。<x:set>标签的使用语法如图7.64所示。



7.5.2 流程控制标签

- * XML流程控制标签包含<x:if>、<x:choose>、<x:when>、<x:otherwise>及<x:forEach>标签。这几个标签的功能和用法和核心标签库对应的标签非常类似，所以我们就在此就不再赘述了，读者可以对照核心标签库的内容来体会流程控制标签的用法。

7.5.3 转换操作标签

- * XML转换操作标签包含<x:transform>标签和<x:param>标签。<x:param>标签的作用和用法也与<c:param>标签类似，我们不再赘述，下面我们来看<x:transform>标签。
- * 在网络应用程序中，经常需要通过XSL样式表对一篇XML文件执行转换，本节所要介绍的<x:transform>标签就是为了实现这一功能设置的。<x:transform>标签的使用语法如图7.66所示。

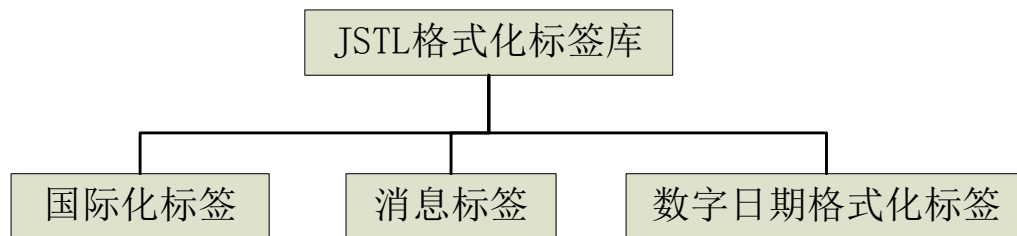
```
<x:transform xml="expression" xslt="expression" var="name"  
    scope="scope" xmlSystemId="expression"  
    xsltSystemId="expression">  
  <x:param name="expression" value="expression"/>
```

7.6 JSTL格式化标签库

- * 格式化标签库又被称为I18N格式标签库，I18N是Internationalization（国际化）的缩写。在不同的国家和地区，对数字和货币等的表示是有所不同的。JSTL提供了格式化标签库来根据不同的语言请求作出不同的响应。

7.6 JSTL格式化标签库

- * 格式化标签库中的标签，根据其功能大致可以分为三类，如图7.67所示。



- * 如果要在JSP页面中使用格式化的标签，需要用taglib指令指明这个标签库的路径为如图7.68所示的形式。

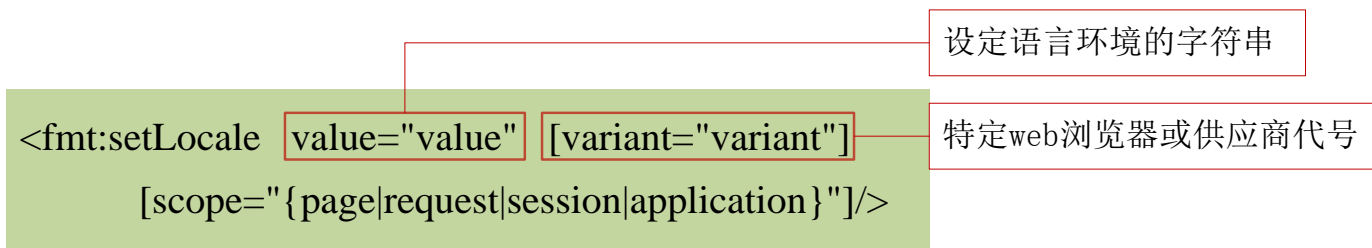


7.6.1 国际化标签

- * 国际化分类标签包含2个标签，一个是设定语言地区的<fmt:setLocale>标签，另一个是设定请求的字符编码集合的<fmt:requestEncoding>标签。

1. <fmt:setLocale>标签

- * <fmt:setLocale>标签的功能是用来设定用户的语言地区代码，即设置格式化或解析日期/时间、数值时所使用的语言环境。<fmt:setLocale>标签的使用格式如图7.69所示。



- * 注意：value属性表示语言地区代码，至于具体的代码是多少，读者可以从<http://www.w3.org/>查询语言代码，在<http://unicode.org>找到国家和地区代码。例如en表示英文、en_US表示英文（美国）、zh_CN表示中文（中国）。

2. <fmt:requestEncoding> 标签

- * <fmt:requestEncoding> 标签用于向JSP容器指定请求(request) 的字符编码，其语法格式如图7.72所示。

```
<fmt:requestEncoding [value="value"] />
```

定义字符编码集合

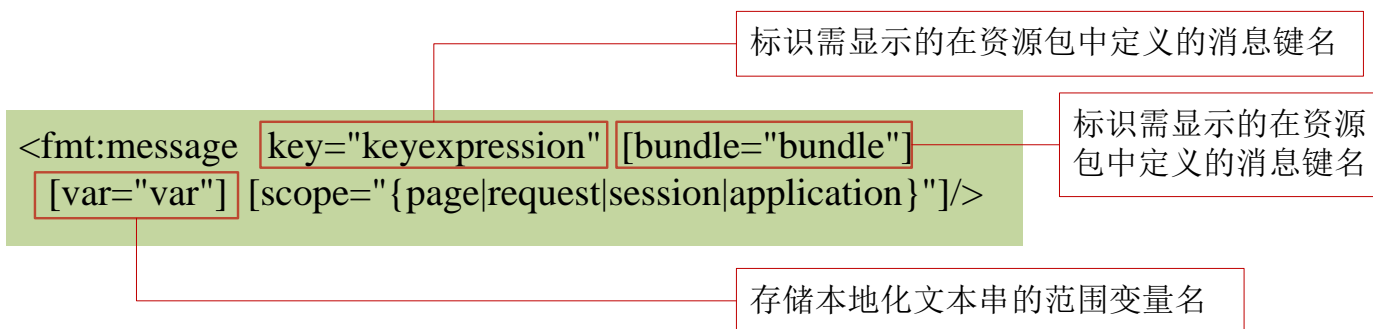
- * <fmt:requestEncoding> 标签就相当于JSP内置对象中的setCharacterEncoding()方法，我们就不再为大家举例了。

7.6.2 消息标签

- * 消息分类标签包含4个标签，分别是<fmt:message>、<fmt:param>、<fmt:bundle>和<fmt:setBundle>。这些标签的主要功能为：获取系统设定的语言资源，从而可以轻易地做到多国化信息。由于这些标签常常是一起使用的，所以我们在对其使用格式进行说明后再统一举例说明它们的用法。

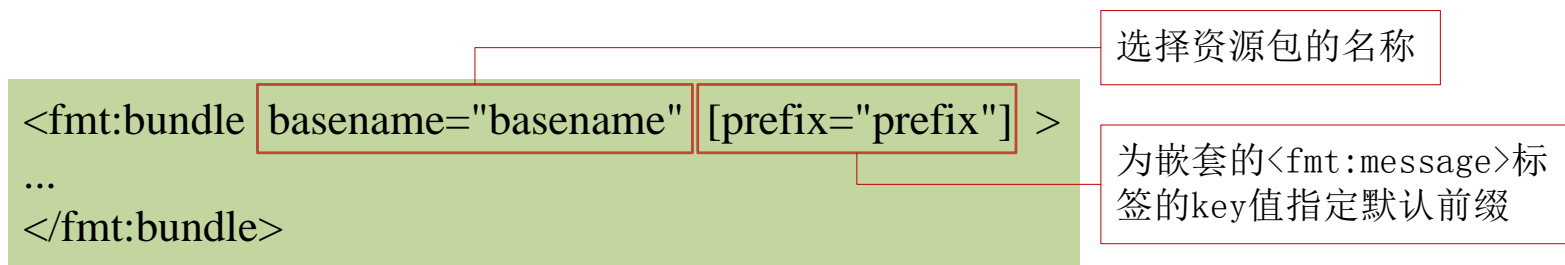
1. <fmt:message>标签

- * <fmt:message>标签用来根据本地化环境从资源包检索文本信息，从而实现文本的本地化。
<fmt:message>标签的使用语法如图7.73所示。



2.<fmt:bundle>标签

- * <fmt:bundle>标签用来根据本地化环境来选择所需的资源包。<fmt:bundle>标签的使用语法如图7.74所示。



3.<fmt:setBundle>标签

- * <fmt:setBundle>标签用来为本地化环境设置一个缺省的资源包，在<fmt:message>标签的特定作用域内起作用。<fmt:setBundle>标签的使用语法如图7.75所示。

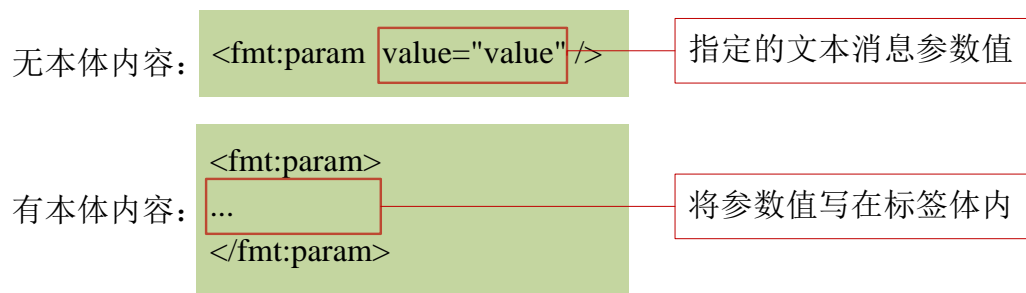
```
<fmt:setBundle basename="basename" [var="var"]  
[scope="{page|request|session|application}"]/>
```

设定语言环境的字符串

设置缺省资源包的范围变量名

4. <fmt:param>标签

- * <fmt:param>标签用来为<fmt:message>标签指定文本消息参数值，动态的设定参数。<fmt:param>标签的使用语法如图7.76所示。



- * 在使用消息标签之前，必须首先创建资源文件，其扩展名为properties，资源文件内容必须按照“key=value”的格式，也就是一个索引对应一个值。资源文件需要放置在应用程序的WEB-INF/classes目录下。

7.6.3 数字日期格式化标签

- * 数字日期格式化分类标签包含6个标签，分别是：
<ftm:setTimeZone>、<ftm:TimeZone>、
<ftm:formatNumber>、<ftm:formatDate>、
<ftm:parseDate>和<ftm:parseNumber>。该类标签的主要功能主要是对页面的数字和日期进行格式化定制。由于篇幅原因，我们在对其使用格式进行说明后再统一举例说明它们的用法。

1. <fmt:timeZone>标签

- * <fmt:timeZone>标签用来设置时区，其使用语法如图7.80所示。

```
<fmt:timeZone value="timezone">  
...  
</fmt:timeZone>
```

设置的时区名

2. <fmt:setTimeZone>标签

- * <fmt:setTimeZone>标签用来将设定了的时区存储在某个范围变量var中，其使用语法如图7.81所示。

```
<fmt:setTimeZone value="timezone" [var="var"]  
[scope="{page|request|session|application}"]/>
```

设置的时区名

存储所设置时区的范围变量

3. <fmt:formatDate>标签

- * <fmt:formatDate>标签用来设定日期和时间的格式并按照设置的格式给予输出，其语法如图7.82所示。

```
<fmt:formatDate value="date/time" [type="type"] [dateStyle="datestyle"]  
[timeStyle="timestyle"] [pattern="patternexpression"] [timeZone="timezone"]  
[var="varname"] [scope="{page|request|session|application}"] />
```

- * <fmt:formatDate>标签的各个属性的具体描述如表7.11所示。

名称	类型	属性描述
value	java.util.Date 类	指定要进行格式化的日期/时间
type	String	指定要设置格式的Date实例的部分，可为time/date/both
dateStyle	String	设置使用于特定语言环境的日期格式，可为default/short/medium/long/full
timeStyle	String	设置使用于特定语言环境的时间格式，可为default/short/medium/long/full
pattern	String	用户自定义日期/时间格式
timeZone	String或java.util.TimeZone	需格式化的时间所在时区
var	String	用来存储格式化后结果的范围变量名
scope	String	范围变量var的作用域

4. <fmt:parseDate> 标签

- * <fmt:parseDate> 标签用于解析日期和时间值，其使用语法如图7.83所示。

无本体内容：

```
<fmt:parseDate value="data/time"[type="type"] [dateStyle="datestyle"]  
[timeStyle="timestyle"] [pattern="patternexpression"] [timeZone="timezone"]  
[parseLocale="parselocale"] [var="var"] [scope="{page|request|session|application}"] />
```

有本体内容：

```
<fmt:parseDate [type="field"] [dateStyle="datestyle"] [timeStyle="timestyle"]  
[pattern="patternexpression"] [timeZone="timezone"] [parseLocale="parselocale"]  
[var="var"] [scope="{page|request|session|application}"] >
```

body content

```
</fmt:parseDate>
```

4. <fmt:parseDate>标签

* <fmt:parseDate>标签的各个属性的具体描述如表7.12所示。

名称	类型	属性描述
value	java.util.Date 类	指定要进行解析的日期/时间
type	String	指定需要解析的Date实例的部分，可为time/date/both
dateStyle	String	设置Date实例日期部分解析方式，可为default/short/medium/long/full
timeStyle	String	设置Date实例时间部分解析方式，可为default/short/medium/long/full
pattern	String	用户自定义日期/时间的格式以确定解析方式
timeZone	String或java.util.TimeZone	需解析的时间所在时区
parseLocale	String或为java.util.Locale类	指定一种语言环境，根据这种语言环境来解析日期/时间值
var	String	用来存储解析结果的范围变量名
scope	String	范围变量var的作用域

5. <fmt:formatNumber> 标签

- * <fmt:formatNumber> 标签用于格式化数值，即设置特定语言环境下的数值的输出方式，其使用语法如图 7.84 所示。

```
<fmt:formatNumber value="number"[type="type"] [pattern="patternexpression"]  
[currencyCode="currencycode"][currencySymbol="currencysymbol"]  
[maxintegerDigits="maxintegerdigits"][minintegerDigits="minintegerdigits"]  
[maxFractionDigits=" maxfractiondigits"][minFractionDigits=" minfractiondigits"]  
[groupingUsed="groupinUsed"][var="var"] [scope="scope"]/ >
```


5. <fmt:formatNumber>标签

* <fmt:formatNumber>标签的属性描述如表7.13所示。

名称	类型	属性描述
value	String或为Number	指定需格式化的数值
type	String	指定格式化方式，其取值有number、currency、percent
pattern	String	用户自定义格式化方式
currencyCode	String	设置所显示数值的货币单位
currencySymbol	String	设置所显示数值的货币符号
maxintegerDigits	int	设置格式化后数值的最大整数位数
minintegerDigits	int	设置格式化后数值的最小整数位数
maxFractionDigits	int	设置格式化后数值的最大小数位数
minFractionDigits	int	设置格式化后数值的最小小数位数
groupingUsed	Boolean	指定格式化后是否要对小数点前面的数字分组
var	String	用来存储格式化后数值的范围变量名称
scope	String	范围变量var的作用域

6. <fmt:parseNumber> 标签

- * <fmt:parseNumber> 标签用来解析数值字符串，其使用语法如图7.85所示。

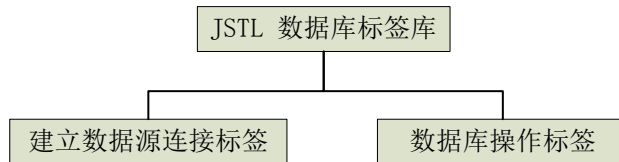
```
<fmt:parseNumber value="numberstring"[type="type"]  
[pattern="patternexpression"][parseLocale="parseLocale"]  
[integerOnly="integerOnly"][var="var"] [scope="scope"]/>
```

- * <fmt:parseNumber> 标签的属性描述如表7.14所示。

名称	类型	属性描述
value	String	需解析的数值字符串
type	String	指定数值字符串，其取值有number、currency、percentage
pattern	String	用户自定义数值字符串的格式以确定解析方式
parseLocale	String或为java.util.Locale类	指定一种语言环境，根据这种语言环境来解析数值字符串
integerOnly	Boolean	设置要解析的部分是否仅为数值字符串的整数部分
var	String	存储数值解析结果的范围变量名
scope	String	范围变量var的作用域

7.7 JSTL数据库标签库

- * 数据库开发在JSP中占有非常重要的地位，JSTL也提供了对数据库操作的支持，通过是使用JSTL，数据库操作可以简化为简单的几个标签，大大提高了数据库开发的效率和程序的可维护性。JSTL的数据库操作标签库中的标签，根据其功能大致可以分为两类，如图7.88所示。



- * 如果要在JSP页面中使用数据库标签库的标签，需要用taglib指令指明这个标签库的路径为如图7.89所示的形式。



7.7.1 建立数据源连接标签

* 建立数据源连接分类标签只包含一个标签

`<sql:setDataSource>`，该标签用来建立数据库连接。

`<sql:setDataSource>`标签的使用语法如图7.90所示。

使用已存在的数据
源连接数据库

```
<sql:setDataSource dataSource="dataSource"  
[var="varName"]  
[scope="{ page|request|session|application}"/> />
```

已存在的数据源

存储数据库连接的变量

使用JDBC方式
连接数据库

```
<sql:setDataSource  
driver="driverName"  
url="URL"  
[user="userName"]  
[password="password"]  
[var="varName"]  
[scope="{ page|request|session|application}"/> />
```

数据库驱动程序字符串

数据库连接字符串

数据库登录用户名

数据库登录密码

7.7.2 数据库操作标签

- * 数据库操作分类标签包含5个标签，分别是：
<sql:query>、<sql:update>、<sql:param>、
<sql:dataParam>和<sql:transaction>。

1. <sql:query>标签

- * <sql:query>标签的功能是执行数据库中的查询操作。
<sql:query>标签的使用语法如图7.92所示。

无本体内内容:

```
<sql:query var="varName" dataSource="dataSource"  
[maxRows="max"] [startRow="start"]  
[scope="{page|request|session|application}"] />
```

maxRows和startRow
可以控制取出结果集
的大小和开始位置

有本体内内容:

```
<sql:query sql=" SQL语句" var="varName" dataSource="dataSource"  
[maxRows="max"] [startRow="start"]  
[scope="{page|request|session|application}"]>  
本体内内容  
</sql:query>
```

2. <sql:param>标签

- * <sql:param>标签的功能就是向<sql:query>标签的SQL语句中传递参数，该标签的使用语法如图7.93所示。

无本体内容：

```
<sql:param value="value" />
```

参数值由value属性指定

有本体内容：

```
<sql:param>value</sql:param>
```

参数值由本体内容指定

3. <sql:update>标签

- * <sql:update>标签的功能是对数据库进行插入、更新和删除操作。<sql:update>标签的使用语法如图7.94所示。

无本体内容:

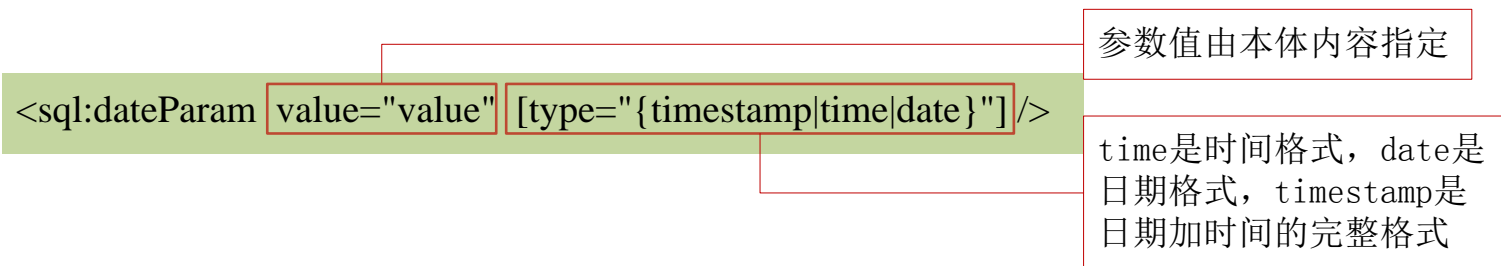
```
<sql: update sql="SQL语句" dataSource="dataSource" [ var="varName"]  
[scope="{page|request|session|application}"] />
```

有本体内容:

```
<sql: update dataSource="dataSource" [var="varName"]  
[scope="{page|request|session|application}"]>  
SQL语句  
</sql: update >
```


4. <sql:dateParam> 标签

- * <sql:dateParam> 标签和 <sql:param> 标签的功能和用法完全相同，不同之处是 <sql:dateParam> 标签是用来设置日期格式的参数，其使用语法如图 7.95 所示。



5. <sql:transaction>标签

- * 在JSTL中，同样支持数据库操作的事务处理，在JSTL中是采用<sql:transaction>标签来实现这个功能的。<sql:transaction>标签的使用语法如图7.96所示。

```
<sql:transaction [dataSource="dataSource"]  
  [isolation="{read_uncommitted|read_committed|repeatable_read|serializable}"] >  
  数据库操作  
</sql:transaction>
```

指定事务隔离的级别，若未指定，则按照用户数据源的事物隔离级别

7.8 JSTL函数标签库

- * JSTL函数标签库就是一些常用的函数，在JSTL中把这些常用的函数封装成标签的形式，然后可以在JSP页面上进行方便的调用。如果要在JSP页面中使用函数标签库的标签，需要用taglib指令指明这个标签库的路径为如图7.98所示的形式。



7.8 JSTL函数标签库

- * 限于篇幅，本节以表格的方式列出每个函数标签的语法和功能，而不再对每一个函数的参数作具体的讲解，如表7.15所示。有兴趣的读者可参阅有关书籍。

函数	描述
<code>fn:contains(string, substring)</code>	如果参数string中包含参数substring，返回true
<code>fn:containsIgnoreCase(string, substring)</code>	如果参数string中包含参数substring（忽略大小写），返回true
<code>fn:endsWith(string, suffix)</code>	如果参数string以参数suffix结尾，返回true
<code>fn:escapeXml(string)</code>	将有特殊意义的XML(HTML)转换为对应的XML character entity code，并返回
<code>fn:indexOf(string, substring)</code>	返回参数substring在参数string中第一次出现的位置
<code>fn:join(array, separator)</code>	将一个给定的数组array用给定的间隔符separator串在一起，组成一个新的字符串并返回
<code>fn:length(item)</code>	返回参数item中包含元素的数量。参数Item类型是数组、collection或者String。如果是String类型,返回值是String中的字符数
<code>fn:replace(string, before, after)</code>	返回一个String对象。用参数after字符串替换参数string中所有出现参数before字符串的地方，并返回替换后的结果
<code>fn:split(string, separator)</code>	返回一个数组，以参数separator为分割符分割参数string，分割后的每一部分就是数组的一个元素
<code>fn:startsWith(string, prefix)</code>	如果参数string以参数prefix开头，返回true
<code>fn:substring(string, begin, end)</code>	返回参数string部分字符串，从参数begin开始到参数end位置，包括end位置的字符
<code>fn:substringAfter(string, substring)</code>	返回参数substring在参数string中后面的那一部分字符串
<code>fn:substringBefore(string, substring)</code>	返回参数substring在参数string中前面的那一部分字符串
<code>fn:toLowerCase(string)</code>	将参数string所有的字符变为小写，并将其返回
<code>fn:toUpperCase(string)</code>	将参数string所有的字符变为大写，并将其返回

7.9 小结

- * 本章介绍了EL表达式语言和JSTL标准标签库的基本知识，包括EL简介和EL应用，以及JSTL核心标签库、XML标签库、国际化标签库和数据库操作标签库等。本章的重点和难点都是能否熟练使用JSTL标准标签库，尤其是核心标签库进行编程和开发。读者在实际的开发过程中可以参考本章的示例程序，在学习本章知识的基础上熟练运用JSTL提高开发速度和质量，并在自己的开发中尝试JSTL标签库来简化开发的代码量。

第8章 Struts 2框架入门

- * Struts这个名字来源于在建筑和旧式飞机中使用的支持金属架。它是第一个实现了Web层MVC架构的开源框架。本章我们在简要介绍MVC模式和Struts 2框架安装基础上，实现我们第一个HelloWorld程序的配置与实现。

8.1.1 Struts 2的由来

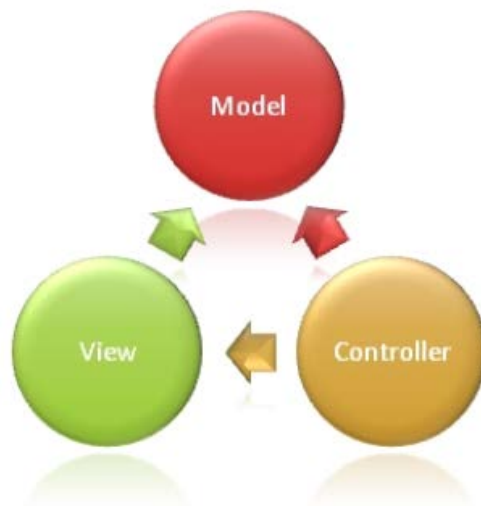
- * Struts 2是Struts的下一代产品，是在Struts和WebWork的技术基础上进行了合并的全新的Struts 2框架，如图8.1所示。



- * 但是Struts 2的体系结构与Struts 1的体系结构的差别巨大。Struts 2是以WebWork为核心的，所以Struts 2可以理解为WebWork的更新产品。但是由于Struts 1名声较大的缘故，所以合并之后Apache基金会将其命名为Struts 2。
- * Struts 2是一个基于J2EE平台的MVC框架，它主要是采用Servlet和JSP技术来实现的。下面我们就带领大家一起来进入Struts 2框架的世界。

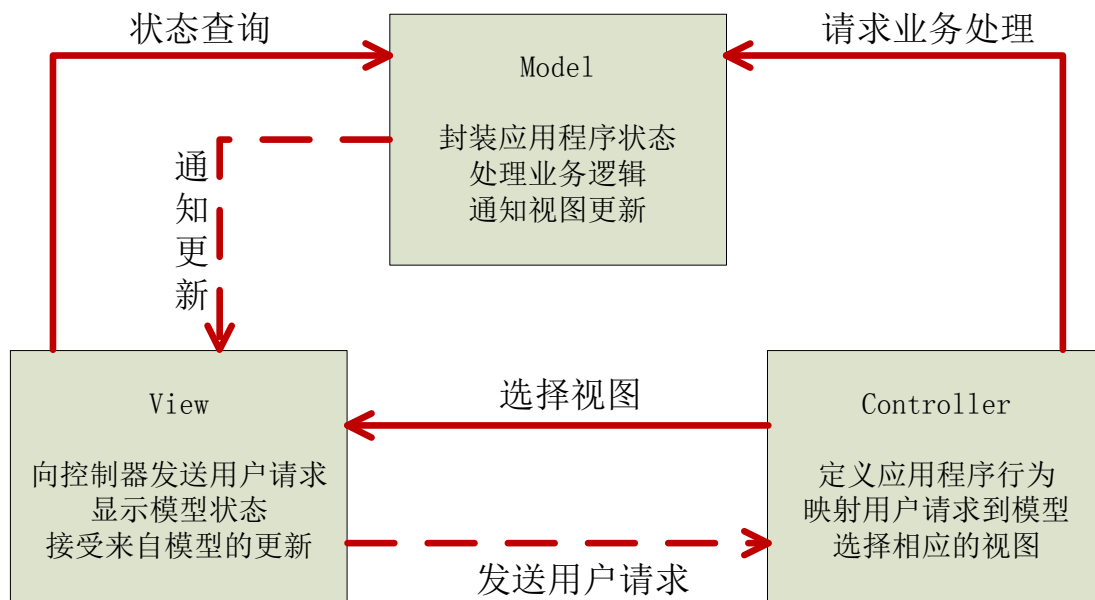
8.1.2 MVC模式

- * MVC是一种设计模式，最早是由Xerox（施乐）公司在20世纪80年代提出的。随后，它成为了一种著名的用户界面设计架构，如图8.2所示。



8.1.2 MVC模式

- * MVC英文全称为Model-View-Controller，即把一个应用程序的输入层、业务处理层、控制流程层按照View、Model、Controller的方式实现了分离，并分别承担不同的任务。图8.3显示了这三个模块各自的功能。

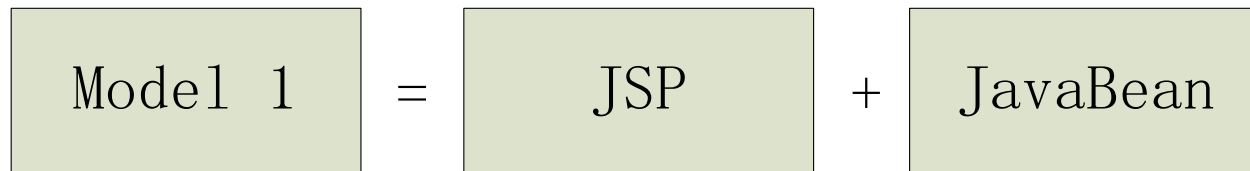


8.1.2 MVC模式

- * MVC模式体现了分层设计的思想，它有以下几点好处：
- * 从视图方面来说，由于多种视图可共享一个后台模型，这就为实现多种用户界面提供了便利。
- * 从模型方面来说，由于其实现与界面独立，因此模型只需提供接口供上层调用，很好的体现了面向对象设计的信息封装和隐藏的原则。
- * 从控制器方面来说，控制器作为介于视图和后台模型间的控制组件，可更好的维护程序流程，选择业务模型，选择用户视图，使程序的调用规则更加清晰，很大程度上优化了系统结构。
- * 正是由于MVC的优势，使它成为软件设计的典范，目前大多数系统都采用了MVC模式来进行系统架构与实现。

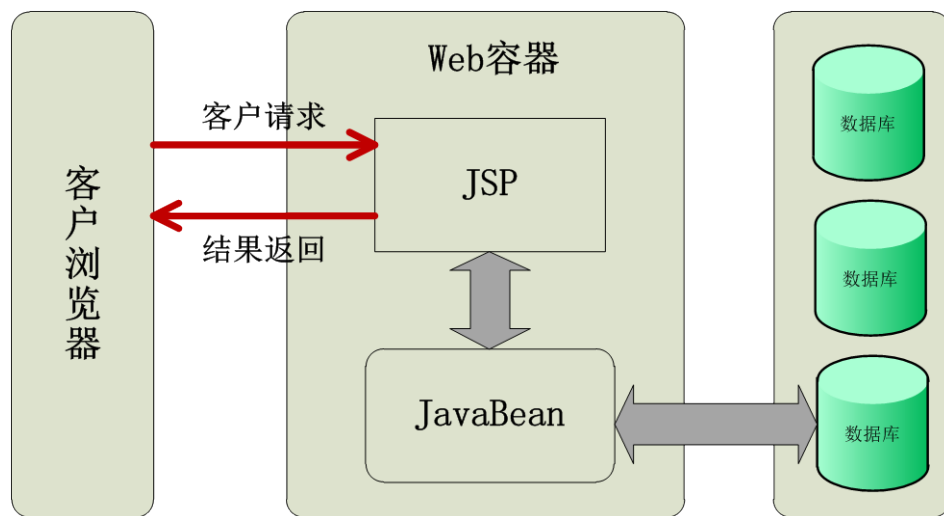
8.1.3 Java Web的实现模型

- * 在Java Web领域存在着两种经典模型，也可以称为实现模式，分别是Model 1和Model 2。这两种模型都是由Sun公司提出的，它们都可被看作是MVC的具体实现形式。现在我们就来比较一下这两种模型。首先来看Model 1，如图8.4所示。



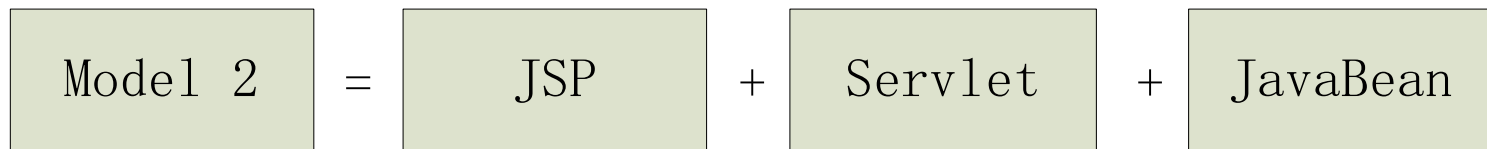
8.1.3 Java Web的实现模型

- * 在这种模型中，JSP充当着控制器与视图的双重角色，JavaBean扮演了模型的解色。JSP直接调用后台模型进行业务处理，同时，再由JSP返回用户结果界面，如图8.5所示。



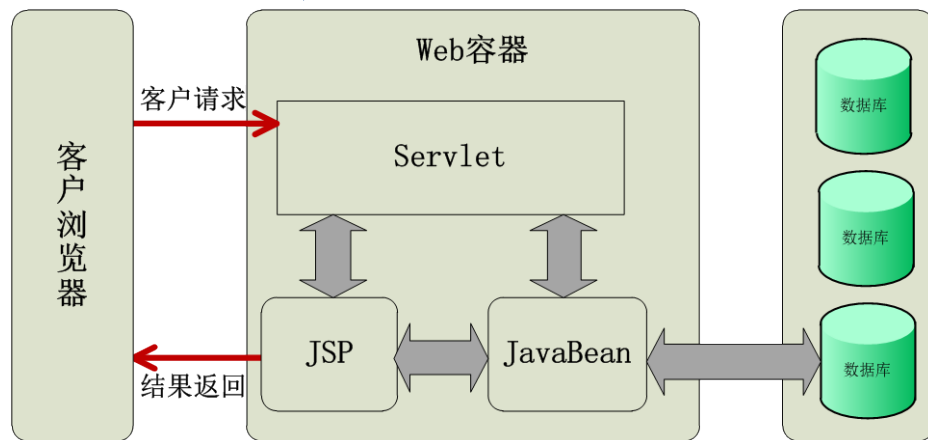
8.1.3 Java Web的实现模型

- * 这种模型对于一些小型的程序还是可以满足的，但对于大规模的系统就显得有些力不从心。因为倘若将JSP既当成控制器又当成视图，那么，在页面代码里就会有大量的HTML标记与Java语言的混合物，这对程序的维护是非常不利的，而且对于页面开发人员和程序设计人员的分工将造成太多约束，无法使它们并行工作，开发效率也就被大大降低。所以Sun公司在Model 1基础上开发出了Model 2模型，如图8.6所示。



8.1.3 Java Web的实现模型

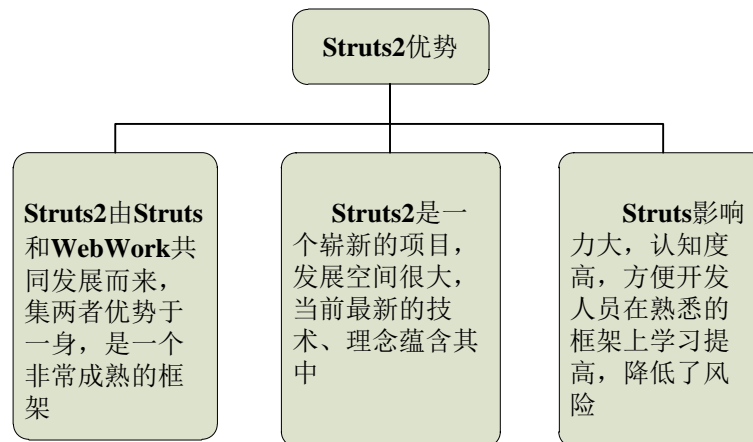
- * 在Model 2模型中，JSP既作为视图又作为控制器的局面不再存在了，而是使用了Servlet作为控制器，JSP则单纯的只负责显示逻辑（还包括很少量的Java代码），如图8.7所示。



- * Model 2清楚的划分了表达、控制、模型这三层结构，很好的实现了MVC设计思想。因此，对于大型系统的设计与开发Model 2提供了很大的帮助。

8.1.4 为什么要使用Struts 2

- * Struts 2是目前最为成功的J2EE框架之一，在众多的MVC框架之中脱颖而出，受到了绝大部分程序员的青睐。究其原因，是因为Struts 2具备了其他框架无法比拟的优势，如图8.8所示。



- * Struts 2有着如此巨大的优势，相信大家一定有了跃跃欲试的感觉，下面我们就知道大家如何在自己的计算机中完成Struts 2的安装和配置。

8.2 Struts 2的下载与安装

- * 本节我们为大家介绍如何下载和安装Struts 2，以及Struts 2中包含文件的作用，然后通过Struts 2自带的实例验证Struts 2安装是否成功。

8.2.1 Struts 2的下载过程

- * Apache官方网站提供最新版本的Struts 2下载，所以建议读者到官方网站下载。下面将详细讲解Struts 2的下载过程。
- * 在浏览器地址栏中输入Apache struts官方网站网址<http://struts.apache.org/>。页面更新后单击Recent Release模块中的Struts 2.3.4版本进行下载。
- * 注意：Struts 1与Struts 2仍然在同一个网页中，没有明显区别，下载时读者要特别注意版本号。即版本号要以2开头。
- * 在进入的下载页面中，我们选择Full Releases版本集合进行下载。单击Struts 2.3.4中的完全发布版（Full Distribution）struts-2.3.4-all.zip版本完成Struts 2的下载。

8.2.1 Struts 2的下载过程

- * 下载完成后，我们得到一个zip文件，将其进行解压，可以看到该文件夹中包括4个目录。它们各自具有不同的作用，如图8.11所示。

src源代码目录	存放*. java文件，Struts 2是一个开源项目，可在此存放所有的代码
doc文档目录	存放文档文件
lib库目录	存放提供给开发人员的jar文件，开发过程中需要将其加入到CLASSPATH中
apps为例子目录	存放Struts 2给出的实例，都是*. war文件

8.2.2 Struts 2安装过程

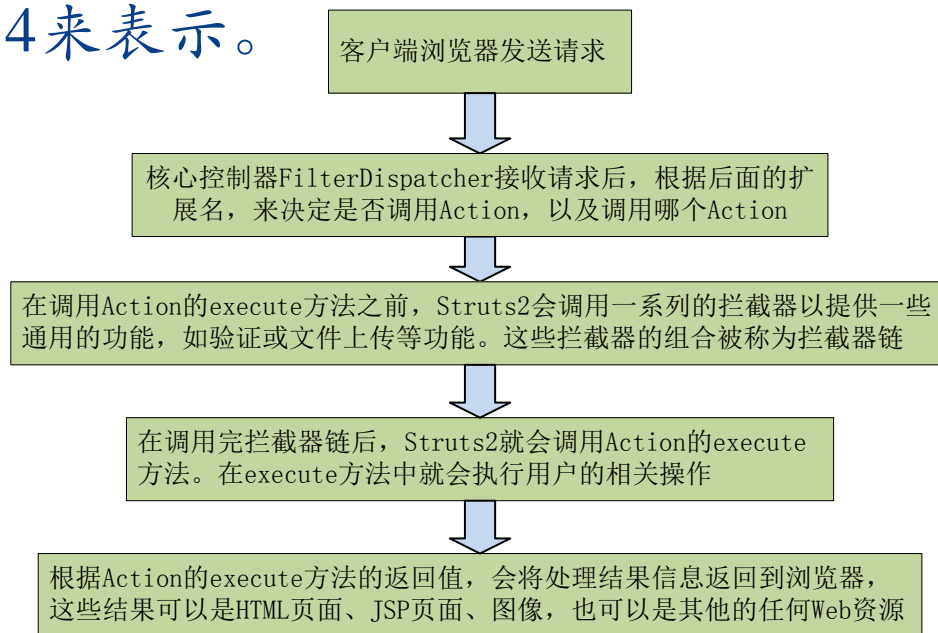
- * 首先我们在MyEclipse软件中新建一个Struts项目工程，在Struts工程的右键菜单上单击【Properties】命令，打开【Properties for Struts2】对话框，在Struts 2解压后的lib目录中选择如下7个文件：
 - * struts2-core-2.3.4.jar
 - * xwork-core-2.3.4.jar
 - * ognl-3.0.5.jar
 - * freemarker-2.3.19.jar
 - * commons-logging-1.1.1.jar
 - * commons-fileupload-1.2.2.jar
 - * commons-lang3-3.1.jar

8.3 使用Struts 2实现第一个程序

- * 这一节我们为大家介绍如何利用Struts 2进行Web应用开发。首先我们学习一下Struts 2的工作流程。

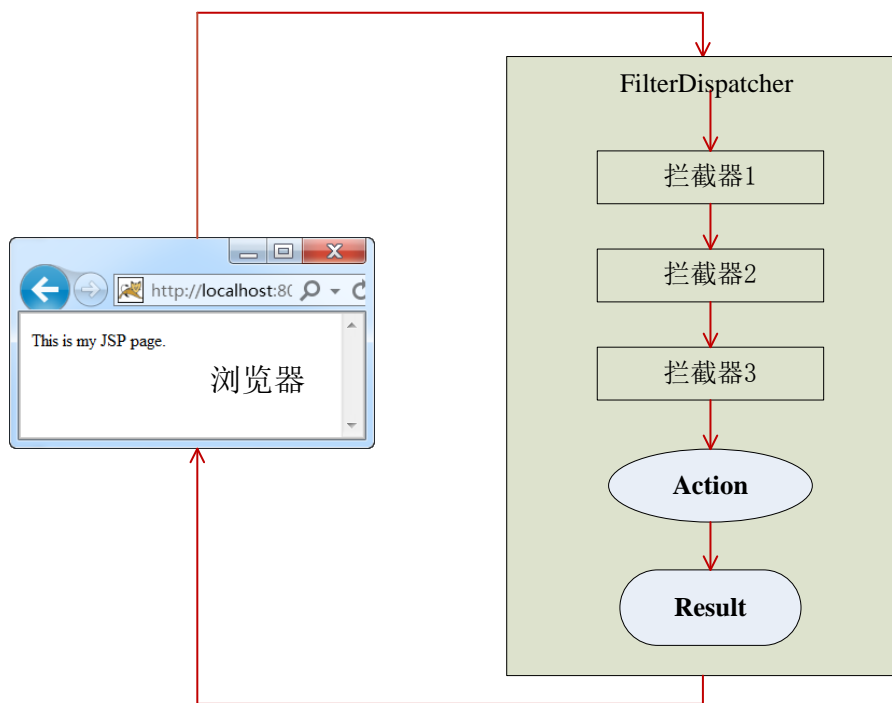
8.3.1 Struts 2的工作流程

- * Struts 2与WebWork的工作方式类似，Struts 2同样使用了拦截器作为其处理用户请求的控制器。在Struts2中有一个核心控制器FilterDispatcher，它负责处理用户的所有请求，如果遇到以.action结尾的请求URL，就会交给Struts 2框架来处理。Struts 2的工作流程我们可以用图8.14来表示。



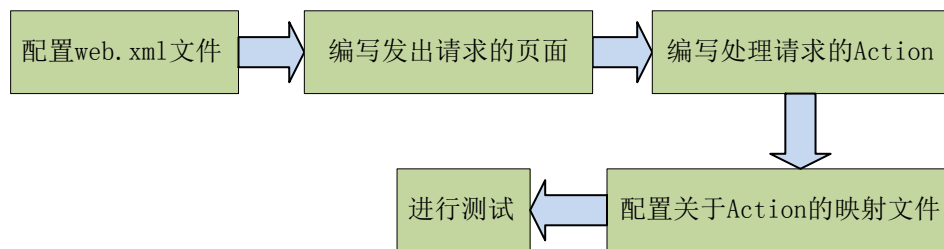
8.3.1 Struts 2的工作流程

- * 我们还可以对图8.14的内容简化为如图8.15所示的样式。



8.3.2 开发一个Struts 2框架程序的步骤

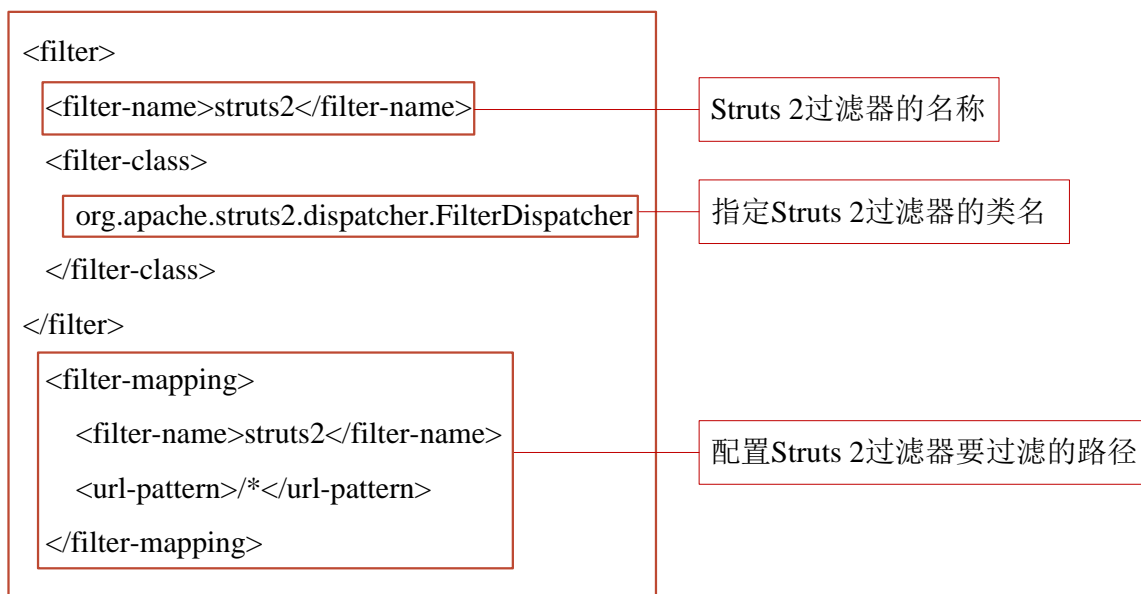
- * 对应于Struts 2的工作流程，我们来为大家讲解一下开发一个Struts 2框架程序的步骤，其步骤大致如图8.16所示。



- * 知道了Struts 2框架程序的开发步骤，我们就可以开始Struts 2程序的开发了，同样，我们来看看如何用Struts 2输出最经典的“HelloWorld!”语句。

8.3.3 配置web.xml

- * Struts 2的web.xml文件配置方法非常简单，即在web.xml中配置Struts 2提供的过滤器，并设置为所有的请求（/*）都要通过这个过滤器，如图8.17所示。



8.3.4 编写JSP界面

- * 接下来我们编写一个页面文件。在这个例子中使用了Struts 2标签库提供的“property”标签用来显示message的属性值。

8.3.5 编写Action

- * Action类是最基本的逻辑处理单元，在MVC模式中分发器分发给不同的Action类，来处理请求。在Struts 2中Action类不必再实现Action接口，可以是任何类。但是一般还要继承ActionSupport类，因为它提供了大量的基本功能，如错误信息处理等。

8.3.6 配置文件中增加映射

- * Struts 2的配置文件是struts.xml，所有请求和分发以及其他配置都在这个文件中定义，struts.xml文件应该放在WEB-INF目录下的classes文件中。如示例8.3所示，配置了一个名称为HelloWorld的action，处理类是struts2.HelloWorld，处理后的结果转到helloWorld.jsp页面上。

8.4 小结

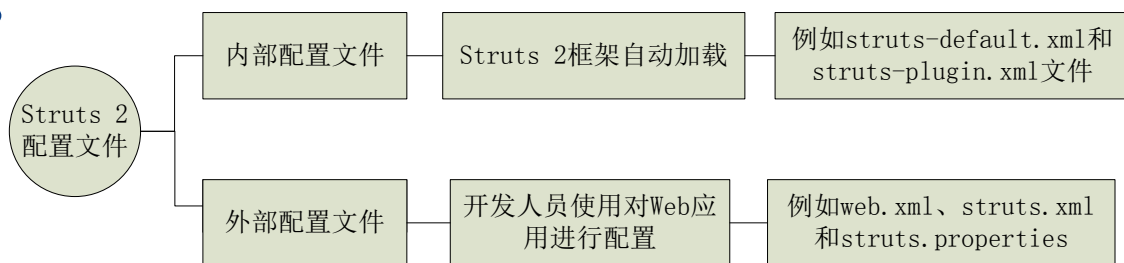
- * 本章是Struts 2的入门章节，主要介绍了Struts 2的一些基础知识及下载安装过程。最后我们又通过一个HelloWorld程序展示了Struts 2框架程序的开发步骤。虽然本章内容不多，但是是后面所有Struts 2开发框架学习的基础。读者应多加练习，熟练掌握Struts 2程序的开发步骤和配置方法。

第9章 Struts 2配置详解

- * 在Struts 2框架中，常用的配置文件包括web.xml、struts.xml、struts.properties和struts-default.xml等。其中struts.xml是Struts 2框架的核心配置文件，负责管理Struts 2框架的业务控制器Action和拦截器等。本章将详细介绍Struts 2应用中的各种配置文件，掌握这些配置文件后才能更好地使用和扩展Struts 2框架的功能。

9.1 Struts 2配置文件

- * Struts 2框架的配置文件可以分为两类：内部配置文件和开发人员使用的配置文件。其具体内容如图9.1所示。

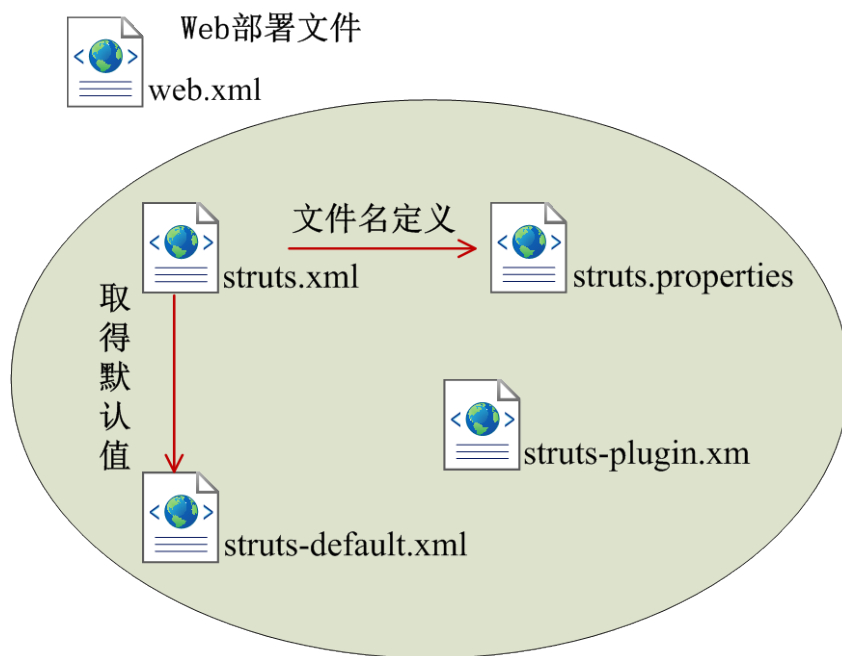


- * Struts 2框架的主要配置文件如表9.1所示。

配置文件	必选	位置（相对于webapp）	说明
web.xml	是	/WEB-INF/	Web部署描述文件，包括所有的必需框架组件
struts.xml	否	/WEB-INF/classes	Struts 2主要配置文件
struts.properties	否	/WEB-INF/classes	Struts 2框架的属性配置文件
struts-default.xml	否	/WEB-INF/lib/struts2-core-x.x.x.jar	Struts 2框架提供的默认配置
struts-plugin.xml	否	/WEB-INF/lib/struts2-xxx-plugin.jar	Struts 2框架的插件配置文件

9.1 Struts 2配置文件

* 这几个配置文件的相互关系如图9.2所示。



9.1.1 web.xml 文件

- * Web应用都需要一个配置文件web.xml，该文件用来对整个应用程序进行配置。在不同的Web框架中，web.xml文件是不同的。在Struts 2框架中，web.xml文件需要配置一个前端过滤器：
StrutsPrepareAndExecuteFilter，用于对Struts 2框架进行初始化以及处理所有的请求。
- * 其实，StrutsPrepareAndExecuteFilter过滤器还可以进行进一步的配置。

9.1.1 web.xml 文件

- * StrutsPrepareAndExecuteFilter 类的初始化参数由3个，具体描述如表9.2所示。

参数名称	参数描述
config	该参数表示Struts 2框架自动加载的系列配置文件。如果有多个配置文件，中间用英文逗号(,) 分隔
actionPackages	该参数表示Struts 2框架要扫描的包。如果有多个ConfigurationProvider类，中间用英文逗号分隔
configProviders	该参数表示自定义的ConfigurationProvider类，用户可以提供一个或多个实现了ConfigurationProvider接口的类，并将这些类名设置成configProviders属性值。多个类名间用英文逗号分隔

- * 本书所使用的Struts 2为struts-2.3.4.1，所以使用的过滤器为StrutsPrepareAndExecuteFilter，读者可能会在其他书籍中发现有的书使用的是org.apache.struts2.dispatcher.FilterDispatcher过滤器，这是因为FilterDispatcher是struts2.0.x到2.1.2版本的核心过滤器，而StrutsPrepareAndExecuteFilter是自2.1.3版本后就替代FilterDispatcher了。由于StrutsPrepareAndExecuteFilter比FilterDispatcher功能更加齐全，我们建议读者使用新版本的过滤器。

9.1.2 struts.xml 文件

- * struts.xml是Struts 2框架的核心配置文件。struts.xml文件具有重要作用，因为应用中的所有常量、Action和拦截器等几乎都配置在该文件中。
- * struts.xml配置文件中涉及到了Struts 2的DTD（Document Type Definition，文档类型定义）信息。这些存在于struts.xml文件中的DTD是必须的。因为Struts 2在装载struts.xml文件时根据这些DTD信息来核对struts.xml文件中的标签设置是否合法。DTD信息除了有核查功能外，MyEclipse还会根据DTD信息自动列出当前标签允许设置的子标签以及标签的所有属性。
- * 在我们下载的struts-2.3.4.1文件夹中lib文件夹中找到struts2-core-2.3.4.1.jar，将其解压。我们就可以找到其中的struts-2.3.dtd文件，该文件就是struts.xml和struts-default.xml文件的DTD。struts-2.3.dtd文件的部分代码如图9.6所示，其他的限于篇幅读者可以自己理解。

9.1.3 struts-default.xml和struts.properties文件

- * struts-default.xml文件是Struts 2框架的基础配置文件，为框架提供默认配置。struts-default.xml文件是struts2框架默认加载的配置文件。它定义struts2一些核心的bean和拦截器。struts-default.xml文件也是包含在struts2-core-2.3.4.1.jar文件中的。
- * Struts 2框架除了struts.xml配置文件外，还有另外一个核心配置文件，这就是struts.properties。struts.properties文件用于配置Struts 2中所需的大量的属性。

9.1.3 struts-default.xml 和 struts.properties 文件

- * struts.properties 文件是一个标准的属性文件，该文件包含了大量的key-value对，每个key就是一个Struts 2属性，该key对应的value就是Struts 2的一个属性值，我们可以举一个例子如图9.7所示。



9.1.3 struts-default.xml和struts.properties文件

- * struts.properties文件通常放在Web应用的WEB-INF/classes文件夹下，Struts 2框架可以自动加载该文件。
struts.properties文件中常用属性及含义如表9.3所示。

属性名称	含义
struts.configuration	指定加载Struts 2配置文件的配置管理器。默认值org.apache.struts2.config.DefaultConfiguration
struts.locale	指定了Web应用程序默认的locale和encoding scheme，默认值是en_US
struts.i18n.encoding	指定了Web应用程序的默认编码集，正确地设置该属性可以解决客户端请求的中文编码问题
struts.objectFactory	指定了Struts 2默认的ObjectFactory Bean，默认值是spring
struts.custom.properties	指定了Struts 2加载的用户自定义属性文件
struts.velocity.configfile	该属性指定了Velocity框架所使用的velocity.properties文件的位置
struts.custom.i18n.resources	该属性指定了Struts 2所使用的国际化文件，如果有多个资源文件，中间用逗号(,)分隔

- * 注意：Struts 2提供了两种设置Struts 2属性的方式：通过struts.properties以key-value方式来设置Struts 2属性，也可以在struts.xml文件中通过<constant>标签来设置Struts 2的属性。

9.2 struts.xml 文件配置详解

- * 在Struts 2框架的配置文件struts.xml文件中，可以将配置内容分为三大类，其中的每种元素可以包含不同的配置内容，如图9.9所示。

类别名称	包含的配置
管理元素	Bean配置、常量配置、包配置、命名空间配置、包含配置
用户请求处理元素	拦截器配置、Action配置、Result配置
错误处理元素	异常配置

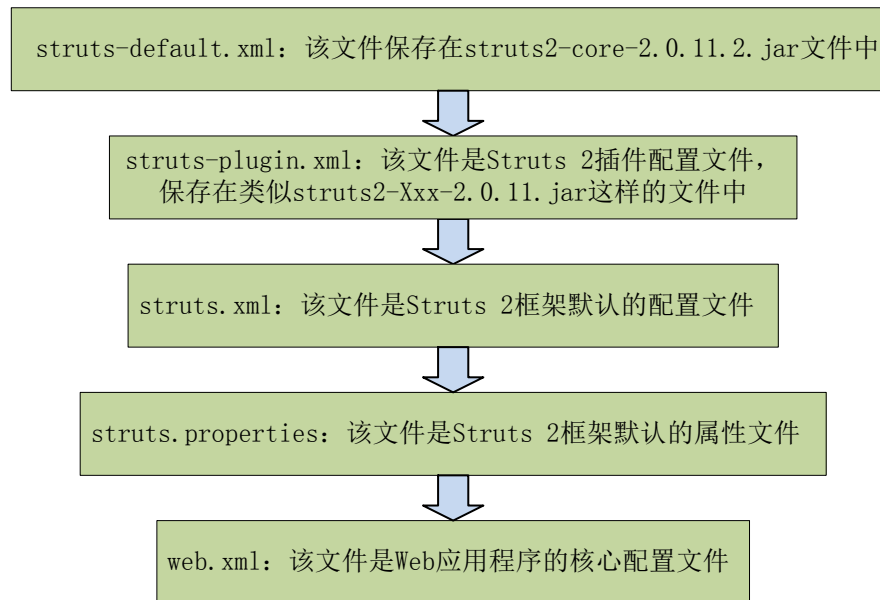
9.2.1 Bean配置

- * 在Struts 2框架的struts-default.xml文件中，定义了大量的核心组件。这些组件不是直接以硬编码的形式写在代码中，而是以自己的依赖注入容器来装配。Struts 2框架可以通过可配置的方式很方便地管理Struts 2的核心组件。
- * 在struts.xml文件中定义的Bean主要有如下两个作用：
- * Struts 2创建该Bean的实例，并作为框架的内部对象使用。
- * 对Bean包含的静态方法进行值注入。
- * struts.xml文件中<bean>元素的属性如表9.4所示。

属性名称	必选	说明
class	是	指定Bean的类名
name	否	指定Bean实例的名字，对于相同类型的多个Bean来说，它们的name属性值必须唯一
type	否	指定Bean实现的接口
scope	否	指定Bean实例的作用域，这个属性值必须是default、singleton、request、session或thread其中之一
optional	否	指定Bean是否为一个可选Bean
static	否	指定Bean是否使用静态方法注入。当指定type属性时，该属性值不应该为true

9.2.2 常量配置

- * 在Struts.xml文件中，通过<constant>元素配置常量，可以作为指定Struts 2属性的一种方式。而Struts 2的常量既可以在struts.xml文件中配置，也可以在struts.properties文件中配置，实际上也可以在其他的一些文件中实现配置。
- * Struts 2框架将按如图9.12所示的搜索顺序加载Struts 2常量。



9.2.2 常量配置

- * 使用<constant>元素配置常量时，需要指定以下两个必填属性。
- * name：该属性指定了常量名。
- * value：该属性指定了常量值。
- * 如果在struts.xml中通过devMode属性来设置Struts 2的工作模式，可以按照如图9.13所示的代码来设置。

```
<struts>
```

```
  <constant name="struts.devMode" value="true" />
```

```
  ... ..
```

```
</struts>
```

设置了**Struts 2**的工作
模式为开发模式

9.2.2 常量配置

- * 在struts.properties文件中的属性和属性值是key-value对，其中key对应于Struts 2常量的name，而value对应于Struts 2常量的value。配置代码如图9.14所示。

```
struts.devMode = true
```

- * 注意：在实际的开发中，最好不要在struts.properties和web.xml文件中配置常量，因为会使代码量明显增加。而Struts 2推荐在struts.xml中配置常量，便于集中管理。

9.2.3 包配置

- * 在Struts 2框架中，其核心组件是Action和拦截器等，该框架使用包来管理这些组件。在包中可以配置多个Action、多个拦截器或者是多个拦截器引用的集合等。使用<package>元素配置包时，可以指定4个属性，如表9.5所示。

属性名	必选	说明
name	是	该属性指定了包的名字。这个名字也是其他包引用的key
extends	否	该属性指定了该包继承的其他包的名字
namespace	否	该属性指定了包的命名空间
abstract	否	该属性指定当前包是否为一个抽象包。抽象包不能包含Action

9.2.4 命名空间配置

- * 命名空间用来解决在同一个Web应用中Action重名的问题。如果在一个Action类中有多个业务处理方法，而客户端请求需要指向不同的方法，这时Struts 2以命名空间（Namespace）的方式来管理Action。
- * 在为包指定命名空间后，在访问包中的Action时就要在Action的name前加上命名空间的名字。如访问edit包中的test动作的URL如下：
- * `http://localhost:8080/web/edit/test.action`
- * 如果将包edit的namespace属性去掉，则可以使用下面的URL来访问test动作：
- * `http://localhost:8080/web/test.action`

9.2.4 命名空间配置

- * 在使用命名空间时要注意以下三点:
- * 命名空间名必须以斜线 (/) 开头, 否则Struts 2不识别该命名空间。
- * 根命名空间和默认命名空间是不同的。如使用 `namespace="/"` 设置了根命名空间后, 如果请求为 `/register.action`, 系统会先在根命名空间中查找register动作, 如果register动作不存在, 则会到默认命名空间中去查找这个Action。
- * Struts 2不会对命名空间分层查找。也就是说, 如果请求为 `/edit/dummy/test.action`, 当命名空间 `/edit/dummy` 下没有test动作时, 系统会直接到默认命名空间去查找test动作, 而不会再到命名空间 `/edit` 去查找test动作。

9.2.5 包含配置

- * 注意：被包含的struts-view.xml、struts-edit.xml等配置文件必须是完整的Struts 2配置文件，也就是说，必须包含DTD信息、配置文件的<struts>标签等信息。一般情况下，将被包含的配置文件和struts.xml都放到WEB-INF\classes目录中。

9.2.6 拦截器配置

- * 拦截器在前面已经多次提到了。拦截器的作用就是在执行Action处理用户请求之前或之后，执行拦截器来进行某些拦截操作。例如，用户请求删除Action时，拦截器判断用户是否具有删除权限，如果没有，则不执行删除操作。

9.3 配置Action

- * Action是Struts 2的核心。在本节将讲解Action的一些常用技术，如Action接口、Action接口的默认实现类ActionSupport、在Action中访问Servlet API，处理多个提交动作和使用通配符等技术。通过本节的学习，读者可以基本了解Action中的常用技术和操作。

9.3.1 Action实现类

- * Struts 2中的Action其实就是一个普通的Java类，在该类中包含一个普通的execute()方法，该方法没有任何参数，返回一个字符串类型的值。
- * 在LoginAction.java中，我们定义了userName属性，用于获取用户提交的表单中的参数值，然后我们可以生成setXXX()和getXXX()方法，分别用来设置和获取属性的值。最后execute()方法返回一个逻辑字符串，例如SUCCESS。
- * 为了使Action类更加规范，Struts 2中提供了一个Action接口。在这个接口中定义了一些常用结果常量，而且在Action接口中还有一个execute方法。如果要实现Action接口，就必须要实现execute方法。

9.3.1 Action实现类

- * 在上述Action接口中，定义了SUCCESS、NONE、ERROR、INPUT和LOGIN这5个常量。这5常量所表示内容如图9.22所示。

常量名称	表示内容
SUCCESS	表示动作执行成功，并应把相应的结果视图显示给用户
NONE	表示动作执行，但不应该吧任何结果视图显示给用户
ERROR	表示动作执行不成功，并应把相应的报错视图显示给用户
INPUT	表示输入验证失败，并应把用户输入的表单重新显示给用户
LOGIN	表示动作没有执行，并应把登录视图显示给用户

- * Action类通常不会实现该接口，而是继承该接口的实现类ActionSupport。ActionSupport实现了非常多的接口，如Validateable、ValidationAware等，并提供了这些接口中方法的默认实现。如果Action类从ActionSupport类继承，将会大大简化Action的开发。

9.3.2 间接访问Servlet API

- * 在Struts 2框架中，Action与Servlet API相分离，这种低耦合性给开发者提供了便利。但是Action不访问Servlet API，就不能实现业务逻辑处理。
- * Struts 2框架认识到了这一点，于是提供了名称为ActionContext的类，在Action中可以通过该类获得Servlet API。创建ActionContext类对象的语法格式如图9.24所示。

```
ActionContext actionContext = ActionContext.getContext();
```

getContext()方法

对象名

9.3.2 间接访问Servlet API

* 在ActionContext类中有一些常用的方法，如表9.6所示。

方法名称	方法描述
Object get(Object key)	通过参数key来查找当前ActionContext中的值，相当于getAttribute(String name)方法
Map getApplication()	返回一个Map类型的ServletContext对象
Map getParameters()	返回一个包含所有HttpServletRequest参数信息的Map对象
Map getSession()	返回一个Map类型的HttpSession对象
static ActionContext getContext()	返回当前线程的ServletContext对象
void put(Object key, Object value)	向当前ActionContext对象中存入键值对信息
void setSession(Map session)	设置一个Map类型的session值

9.3.3 直接访问Servlet API

- * 在9.3.2小节中，通过使用ActionContext类，实现了Action间接访问Servlet API。接下来我们将为大家介绍如何使用Action直接访问Servlet API，直接访问方式可以分为IoC方式和非IoC方式。IoC（Inversion of Control，控制反转）就是将设计好的类交给系统去控制，而不是在自己的内部控制。

1. IoC方式

- * 在Struts 2框架中，通过IoC方式访问Servlet API，就必须在Action中实现相应的接口。这些接口的名称和说明如表9.9所示。

接口名称	说明
ServletContextAware	实现该接口的Action可以直接访问ServletContext对象。Action必须实现该接口的void setServletContext()方法
ServletRequestAware	实现该接口的Action可以直接访问HttpServletRequest对象。Action必须实现该接口的void setServletRequest()方法
ServletResponseAware	实现该接口的Action可以直接访问HttpServletResponse对象。Action必须实现该接口的void setServletResponse()方法
SessionAware	实现该接口的Action可以直接访问HttpSession对象。Action必须实现该接口的void setSession()方法

2. 非Ioc方式

- * 在非Ioc方式中，Struts 2提供ServletActionContext类来获得Servlet API。在ServletActionContext类中也提供了getPageContext()、getRequest()、getResponse()等方法来实现直接访问功能。
- * 注意：对于直接访问Servlet API，我们推荐大家使用非Ioc方式，因为Ioc方式实现起来比较麻烦，并且Servlet API耦合大，而非Ioc方式实现简单，代码量少又能满足要求。

9.3.4 动态方法调用

- * 在业务处理Action中，可以包含一个或者多个逻辑处理方法。例如，JSP文件中的同一个form表单有多个用来提交表单的按钮，当用户通过不同的按钮提交表单时，需要调用Action中的不同处理方法，这时就可以使用动态方法调用。使用动态调用时，form表单的action属性值必须符合如图9.32所示的格式。

```
action = "Action名称! 方法名称"  
action = "Action名称! 方法名称.action"
```

Action名称后要制定要调用的方法名，以“!”连接，可以再加上action后缀

9.3.5 使用method属性和通配符映射

- * 除了name属性和class属性以外，配置Action时还可以使用method属性，其基本格式如图9.38所示。

```
<action name="Action名称" class="包名.Action类名" method="方法名称">  
    <result >视图URL</result>  
</action>
```

- * 注意：在配置中使用通配符时，尽量将<action name="*">的形式放在最后，以防这种形式最先被匹配。

9.3.6 默认Action

- * 在Struts 2框架中，允许用户定义一个默认的动作，当用户请求找不到对应的<action>元素值时，系统将调用默认Action来接收用户请求信息。这个Action也在struts.xml文件中配置，使用的配置元素为<default-action-ref>。

9.4 配置Result

- * 业务控制器Action负责处理用户请求，但它不能提供对用户的直接响应，当处理完请求信息后，需要根据Result结果配置，将Action的处理结果对应到相应的视图。本节就来详细介绍Result配置的相关知识。

9.4.1 结果映射

- * 在前面配置Action时都使用了<result>元素，<result>元素的值可以是JSP页面文件，也可以是一个Action的name值。通过这些配置告诉Struts 2框架，在执行Action后，将响应一个JSP文件或者将执行另一个Action。我们先来看一个最典型的result，如图9.43所示。

```
<action name="register" class="action.RegisterAction">
```

```
<result name="success" type="dispatcher"/>/success.jsp</result>
```

```
</action>
```

通过**type**指定
结果类型

- * <result>元素可以配置在<action>元素中，也可以配置在<action>元素外，根据这两种形式可以将Result的配置分为两类：局部Result和全局Result。

1. 局部Result

- * 局部Result定义<action>元素中，作用范围是这个<action>，这时<result>元素是<action>元素的子元素。

2. 全局Result

- * 全局Result，名称为error。如果default包下的任何一个Action返回字符串ERROR，那么都可以调用这个result，页面将返回error.jsp。
- * 注意：当一个Action的局部Result与全局Result重名时，那么对于该Action的返回视图来说，局部Result会覆盖全局Result。

2. 全局Result

- * 在<result>配置的标准形式中，使用了type属性定义结果映射的类型。默认情况下表示使用JSP视图技术。Struts 2默认支持的视图技术和result类型如表9.10所示。

类型	说明
chain	用于Action链式处理
dispatcher	用来整合JSP，是<result>元素默认类型
redirect	用来重定向到其他文件
velocity	用来整合velocity
xslt	用来整合XML/XSLT
redirectAction	用来重定向到其他Action

- * 下面我们对其中较为重要的几种类型进行讲解。

9.4.2 dispatcher 结果类型

- * dispatcher 结果类型用来表示“转发”到指定结果资源，它是 Struts 2 的默认结果类型。
- * 接下来以添加联系人信息为例，介绍 dispatcher 类型的使用，认识 dispatcher 结果类型下的输出结果。

9.4.3 redirect 结果类型

- * redirect 结果类型用来“重定向”到指定的结果资源，该资源可以使JSP文件，也可以是Action类。

9.4.4 redirectAction结果类型

- * redirectAction结果类型和redirect结果类型相似，也是重定向到其他资源，重新生成一个新的请求。但是在redirectAction结果类型中，使用ActionMapperFactory类的ActionMapper实现重定向。使用redirectAction可以简化对带有命名空间的Action的访问。

9.4.5 使用通配符动态配置result

- * 在配置Action时，可以使用通配符对Action进行动态映射。同样，对<result>元素也可以采用通配符进行动态配置。
- * 这里的<result>元素值不是固定的、唯一的，而是使用{1}.jsp来表示，可以对应多个文件。如果客户端发送book_add.action请求，<result>元素的值将被设置为add.jsp。如果发送book_update.action请求，<result>元素的值将被设置为update.jsp。通过使用通配符，从而实现对请求结果的动态映射。

9.4.6 使用OGNL动态配置result

- * 在上面使用通配符动态配置result时，<result>元素值为{1}.jsp，这是根据URL参数来匹配的。如果根据Action中的属性名进行动态配置，那么就需要使用OGNL表达式。OGNL (bject-Graph Navigation Language) 是一种功能强大的表达式语言，通过它简单一致的表达式语法，可以存取对象的任意属性，调用对象的方法，实现字段类型转化等功能。
- * 在这种配置中，<result>元素值使用OGNL表达式实现。用户输入的联系人为limo，那么这里的<result>元素值就为limo.jsp；如果用户输入的联系人为lester，那么这里的<result>元素值就为lester.jsp。所以为了正确显示页面，我们输入的用户名必须要与创建的JSP文件名保持一致。

9.5 小结

- * 本章是Struts 2的配置章节，是整个Struts 2框架最基础也是最重要的知识。只有正确地理解和熟练地掌握了本章所讲述的知识，我们才能在后面章节的具体应用中的轻松地使用Struts 2。本章的重点内容是web.xml和struts.xml文件的配置、Action文件的配置和Result文件的配置。难点内容是能熟练掌握Action文件和Result文件在项目中的应用。希望大家一定要动手多加练习，熟练掌握本章所讲解的内容。

第10章 Struts 2拦截器

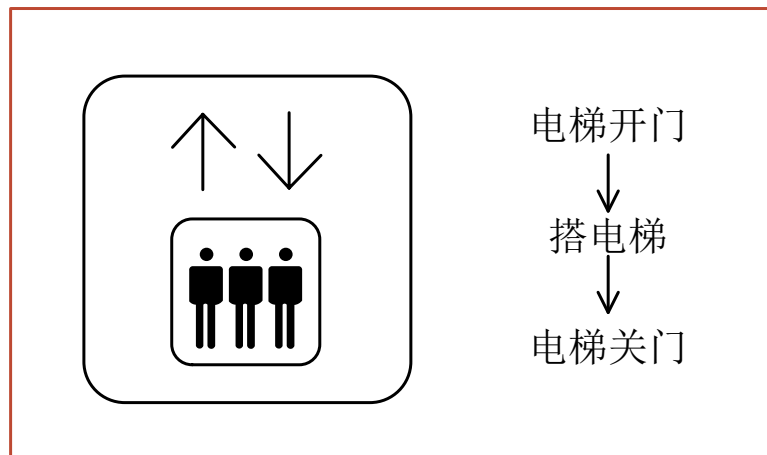
- * 拦截器（Interceptor）是Struts 2框架的核心组成部分。Struts 2的很多功能都是构建在拦截器基础上的，如文件的上传下载、国际化、类型转换等。本章首先对拦截的实现原理和意义进行介绍，然后介绍Struts 2拦截器、如何自定义拦截器，最后通过一个简单的权限拦截器示例具体介绍一下拦截器的应用过程。

10.1 拦截器的实现原理

- * 拦截器顾名思义就是能够拦截用户操作。拦截器的功能就是在进行一个操作（如调用方法）时，它会在用户执行操作前进行一系列操作，同样在用户操作完成后，进行一系列操作。

10.1.1 拦截器简介

- * 到底什么是拦截器呢？我们可以通过一个实际例子来说明。比如现在你要乘坐电梯，在你进入电梯之前，电梯门首先要打开你才能进入。当你进入电梯后，电梯门会自动关闭。这时电梯就可以看做是一个拦截器，在你进入之前和之后都进行了一系列操作，如图10.1所示。

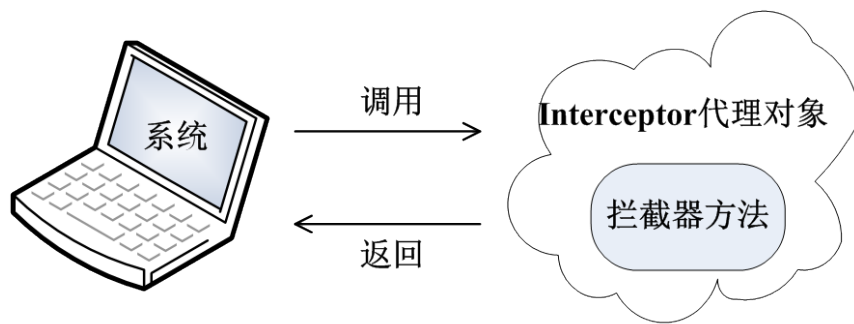


10.1.1 拦截器简介

- * 实际上，拦截器应用了软件开发中的一个重要思想——AOP（Aspect Oriented Programming，面向切面程序设计）。AOP可以看做是OOP（Object Oriented Programming，面向对象程序设计）的补充和完善，它可以将影响多个类的公共行为封装到一个可重用模块。

10.1.2 拦截器实现原理

- * 电梯是自动开门和关门的，那么拦截器是否也有这种功能呢？答案是肯定的。读者可能会有疑问，方法可以在不调用的情况下自动执行吗？其实我们这里所指的自动执行是指通过代码驱动来实现方法的执行的。
- * 所谓的拦截其实就是动态地生成一个代理对象，这个对象中包含了拦截器方法的调用。当调用这个对象时，就会同时调用拦截器的方法，从而实现动态调用拦截器方法的目的。实现过程如图10.2所示。

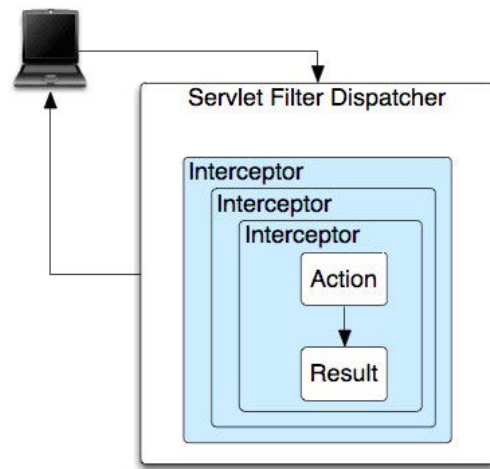


10.2 Struts 2拦截器

- * Struts 2拦截器动态拦截Action调用的对象。它提供了一种机制，使开发者可以定义一个特定的功能模块，这个模块可以在Action执行之前或者之后运行，也可以在一个Action执行之前阻止Action执行。

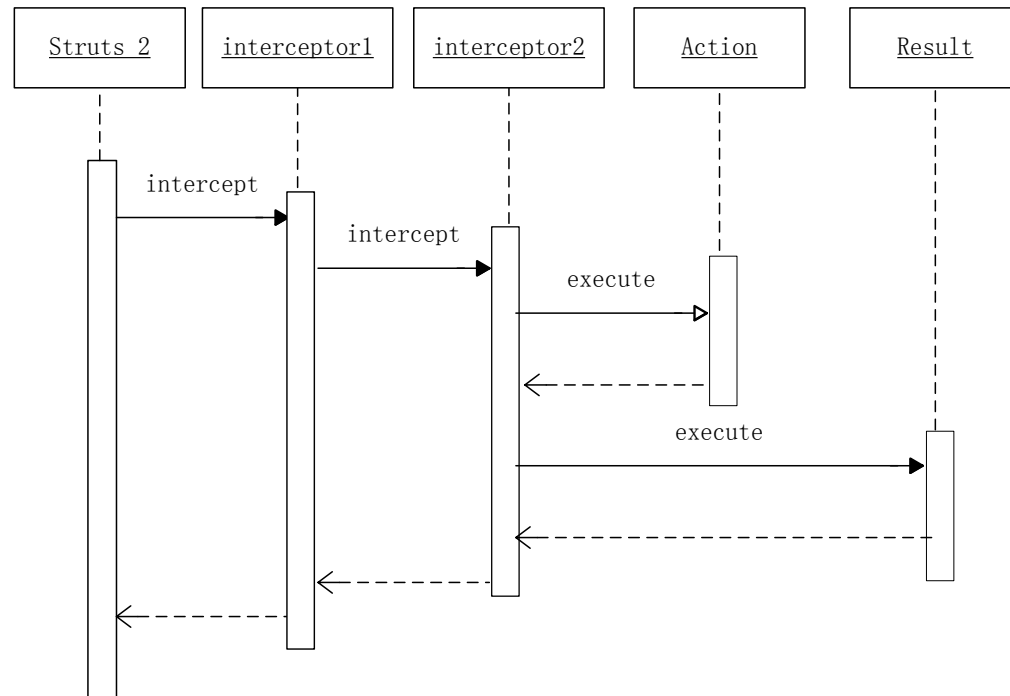
10.2.1 Struts 2 拦截器原理

- * 通常情况下，拦截器都是通过代理的方式调用。当请求到达Struts 2的ServletDispatcher时，Struts 2会查找配置文件，并根据配置实例化相应的拦截器对象，然后将这些对象组成一个列表，最后逐个调用列表中的拦截器，如图10.3所示。



10.2.1 Struts 2 拦截器原理

- * 每个Action请求都包装在一系列的拦截器内部。拦截器可以在Action执行之前做准备工作，也可以在Action之后做回收工作。拦截器的工作时序图如图10.4所示。



10.2.2 配置拦截器

- * 在struts.xml文件中配置一个拦截器非常简单，只需要使用<interceptor>元素指定拦截器类和拦截器名即可。配置拦截器的语法格式如图10.5所示。

```
<interceptor name="拦截器名" class="拦截器对应的Java类" />
```

- * 为了能在多个动作中方便地引用一个或多个拦截器，可以使用拦截器栈将这些拦截器作为一个整体来使用。当拦截器栈被配置到一个Action时，要想执行该Action，必须先执行拦截器栈中的每一个拦截器。拦截器栈的语法格式如图10.6所示。

```
<interceptors>
```

```
    <interceptor-stack name="拦截器栈名">
```

```
        <interceptor-ref name="拦截器名1"/>
```

```
        <interceptor-ref name="拦截器名2" />
```

```
    </interceptor-stack>
```

```
</interceptors>
```

<interceptor-ref>元素指定一个拦截器

10.2.2 配置拦截器

- * 注意：在前面提到的struts-default.xml文件中，系统配置了许多拦截器栈，有兴趣的读者可以去查看。
- * 一个拦截器栈被定义之后，就可以将这个拦截器栈当成一个普通的拦截器来使用，只是在功能上是多个拦截器的有机组合。
- * Struts 2允许开发者将某个拦截器定义为默认拦截器。自定义拦截器需要使用<default-interceptor-ref>元素，配置后，该拦截器将成为所在包中的默认拦截器。

10.3 自定义拦截器

- * 作为一个框架，可扩展性是不可缺少的优点之一。Struts 2框架虽然提供了丰富的内置拦截器，但是仍然允许开发者自定义拦截器。本节就来介绍如何创建和使用自定义拦截器。

10.3.1 自定义拦截器类

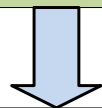
- * 一般来说，在定义一个拦截器类时，需要将该类实现Interceptor接口，或者继承抽象拦截器类AbstractInterceptor。下面我们介绍如何通过继承AbstractInterceptor实现一个自定义拦截器类。

10.3.2 使用自定义拦截器

- * 在实现自定义拦截器之后，如果要使用该拦截器，需要实现如图10.11所示的两个步骤。

1

在struts.xml文件中，对自定义拦截器进行配置



2

在配置Action时，使用<interceptor-ref>元素引入

10.4 Struts 2系统拦截器

- * Struts 2框架的大部分工作都是通过系统拦截器来实现的，例如解析请求参数、参数类型转换等。本节将为大家简要介绍系统拦截器的相关知识。

10.4.1 系统拦截器

- * Struts 2提供了大量的系统拦截器，这些拦截器都是以键值对的形式配置在struts-default.xml文件中的。我们可以从中截取一部分代码如图10.19所示。

```
<interceptor name="alias" class="com.opensymphony.xwork2.interceptor.AliasInterceptor"/>
<interceptor name="chain" class="com.opensymphony.xwork2.interceptor.ChainingInterceptor"/>
<interceptor name="cookie" class="org.apache.struts2.interceptor.CookieInterceptor"/>
<interceptor name="clearSession" class="org.apache.struts2.interceptor.ClearSessionInterceptor"/>
<interceptor name="createSession" class="org.apache.struts2.interceptor.CreateSessionInterceptor"/>
<interceptor name="debugging" class="org.apache.struts2.interceptor.debugging.DebuggingInterceptor"/>
```

- * 如果开发者定义的package继承Struts 2框架的默认包，则可以自由使用上面定义的拦截器，否则只有自己来定义这些拦截器。

10.4.1 系统拦截器

- * Struts 2还利用这些拦截器组合了一系列的拦截器栈，我们截取一部分拦截器栈配置如图10.20所示。

```
-<interceptor-stack name="basicStack"> <interceptor-ref name="exception"/>
  <interceptor-ref name="servletConfig"/>
  <interceptor-ref name="prepare"/>
  <interceptor-ref name="checkbox"/>
  <interceptor-ref name="multiselect"/>
  <interceptor-ref name="actionMappingParams"/>
  -<interceptor-ref name="params"> <param name="excludeParams">dojo\..*,^struts\..*,^session\
  ..*,^request\..*,^application\..*,^servlet(Request|Response)\..*,parameters\...*</param> </interceptor-ref>
  <interceptor-ref name="conversionError"/> </interceptor-stack>
```

基本拦截器栈

```
-<interceptor-stack name="validationWorkflowStack"> <interceptor-ref name="basicStack"/>
  <interceptor-ref name="validation"/>
  <interceptor-ref name="workflow"/>
</interceptor-stack>
```

校验拦截器栈

```
-<interceptor-stack name="fileUploadStack"> <interceptor-ref name="fileUpload"/>
  <interceptor-ref name="basicStack"/>
</interceptor-stack>
```

文件上传拦截器栈

10.4.1 系统拦截器

- * struts-default.xml文件中包括了Struts 2应用所需要的大部分拦截器栈，很多时候只需要使用系统的默认拦截器栈defaultStack就能实现我们所期望的大部分功能。

10.4.2 timer拦截器实例

- * Struts 2系统拦截器中，timer拦截器可以实现输出Action的执行时间，所以也可以称timer拦截器为耗时拦截器。

10.5 权限拦截器实例

- * 在实际应用中，如果用户没有登录系统，则该用户的某些操作将受到限制。如果用户强制进行操作，例如在浏览器地址栏中请求登录后才可以访问URL地址，这时拦截器将执行拦截，重新返回到登录页面。接下来以此为例创建程序，实现用户的登录拦截。

10.5.1 权限拦截器

- * 权限拦截器实际上就是一个实现权限控制的Java类。这个类只需要继承Struts 2框架提供的AbstractInterceptor类，并通过intercept()方法，完成拦截器的权限检查功能。本章主要是说明拦截器的设计思路和实现过程，因此权限检查功能实现的比较简单。
- * 首先是用户登录拦截器的实现类。
- * 这个类主要是实现用户的登录拦截。如果Session中没有登录的用户名，则返回Action.LOGIN；如果用户已经登录，则执行return ai.invoke()，表示继续执行其他操作。

10.5.2 配置拦截器

- * 在创建了登录拦截器后，我们还要在struts.xml文件中进行配置。
- * 在struts.xml文件中要配置的内容有：对拦截器的配置和对Action类的配置。

10.5.3 业务控制器Action

- * 上述配置文件中，已经指出该实例所需要的Action类为LoginAction，该类继承ActionSupport类。在action文件夹下，创建LoginAction.java文件。
- * 这个类比较简单，只是继承ActionSupport类，并实现了execute()方法。
- * 运行程序，在浏览器中输入
<http://localhost:8080/ch10/loginAction.action>。请求登录Action，拦截器起作用，判断session中是否存在用户名信息。如果没有用户名信息就返回登录界面。

10.6 小结

- * 本章主要讲述了Struts 2拦截器的相关知识。拦截器是Struts 2的核心组成部分，Struts 2的大部分功能都是通过拦截器来完成的。本章的重点内容是理解拦截器的实现原理以及自定义拦截器的应用。难点是理解系统拦截器和权限拦截器的运用。希望读者多加练习，争取掌握。

第11章 Struts 2类型转换和输入校验

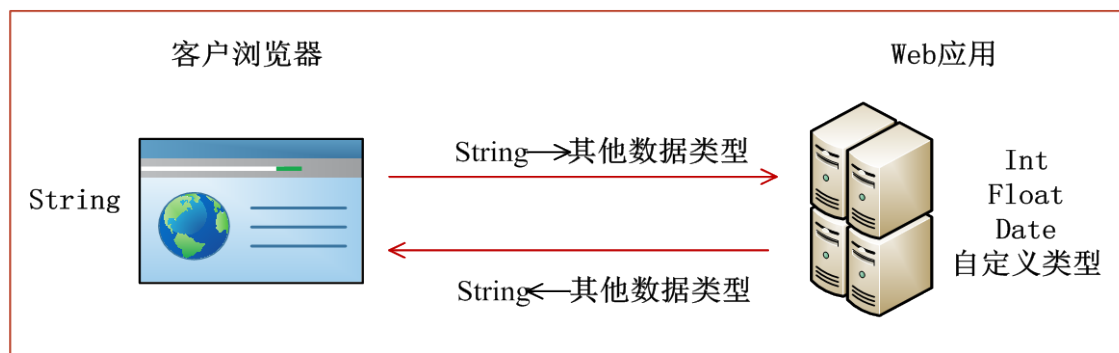
- * 类型转换和输入校验是Struts 2的一个非常重要的部分。通过类型转换能够将表单参数转换成Java类中的各种类型，而通过输入校验能够对表单参数进行合法性判断，从而过滤掉“不合法”数据。本章将详细介绍Struts 2的内建类型转换器和自定义类型转换器，还将介绍如何手动添加校验代码来完成校验和通过校验框架来完成校验。

11.1 Struts 2 类型转换基础

- * 我们都知道在Java中有类型转换，同样在Struts 2中也有类型转换这个概念。不过它不是作用在Java语言内部，而是在表单参数和Java类中的数据中进行的。在所有的基于Web的Java开发框架中，Struts 2拥有最优秀的类型转换能力。

11.1.1 为什么需要类型转换

- * 在Web世界中输入输出是没有数据类型的概念的，任何数据都被当做字符串或字符串数组来传递。而在Web应用的对象中，往往使用了多种不同的类型，如整数（int）、浮点数（float）、日期（Date）或者是自定义数据类型等。因此，在服务器端必须将字符串转换成合适的数据类型；服务器端完成处理后，又要将其他数据类型转换为String，然后传递给客户端进行显示。具体流程如图11.1所示。



11.1.1 为什么需要类型转换

- * Struts 2提供了强大的类型转换功能，对于常用的数据类型提供了内建的数据类型转换器。同样，开发者也可以自行设计类型转换器来进行类型转换。

11.1.2 自定义类型转换器

- * 本小节我们先通过一个实现用户登录的自定义类型转换器例子来了解一下Struts 2的类型转换处理机制。
- * 注意：在页面中输入信息使用的分隔符必须与在Action中定义的分隔符完全一致，包括大小写和中英文状态等。

11.2 使用Struts 2的类型转换

- * 将字符串请求参数转换为相应的数据类型，是所有MVC框架应该提供的功能。Struts 2提供了相当丰富的类型转换功能，从而使开发者一般不再需要建立自定义类型转换器。

11.2.1 内建类型转换器

- * Struts 2为常用的数据类型提供了内建的类型转换器，所以根本不需要使用自定义转换器。对于内建的转换器，Struts 2在遇到这些类型时，会自动去调用相应的转换器进行类型转换。Struts 2提供的主要内建类型转换器如表11.1所示。

数据类型及封装类	转换说明
boolean / Boolean	完成字符串类型和布尔类型之间的转换
char / Character	完成字符串类型和布尔类型之间的转换
int / Integer	完成字符串类型和整数类型之间的转换
float / Float	完成字符串类型和单精度浮点类型之间的转换
long / Long	完成字符串类型和长整数类型之间的转换
double / Double	完成字符串类型和双精度浮点类型之间的转换
Date	完成字符串类型和日期类型之间的转换，日期格式使用当前请求的Locate的SHORT格式

11.2.2 使用集合类型属性

- * 在使用内建类型转换器实现类型转换时，将用户请求参数转换为复合类型的实例对象，但是只实现了添加一条注册信息的效果。如果需要实现同时处理多条注册数据，还需要使用集合类型来实现。

11.3 Struts 2输入校验

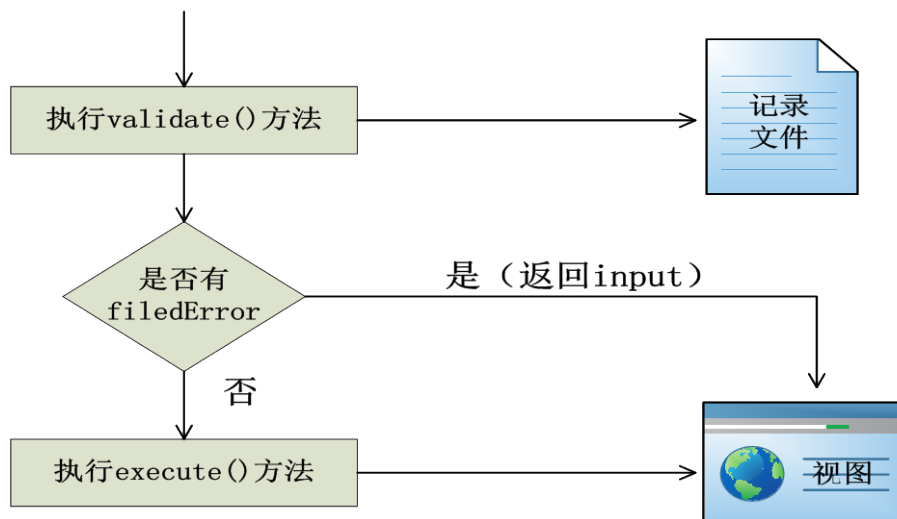
- * 输入校验是所有Web应用必须拥有的功能。因为Web应用的开放性所导致了用户的多样性，因此，用户的输入信息不符合Web系统的要求的可能性非常大。所以就要求Web系统必须具有对用户输入信息的校验功能。Struts 2在一切输入校验之前进行类型转换。类型转化成功后，即对请求信息进行数据校验。

11.3.1 使用validate方法完成输入校验

- * 在Struts 2框架中，`validate()`方法是专门用来校验数据的。具体实现时，可以通过继承`ActionSupport`类，并重写`validate()`方法来完成输入校验。Struts 2框架执行Action类时，会在调用`execute()`方法之前调用该Action类的`validate()`方法。

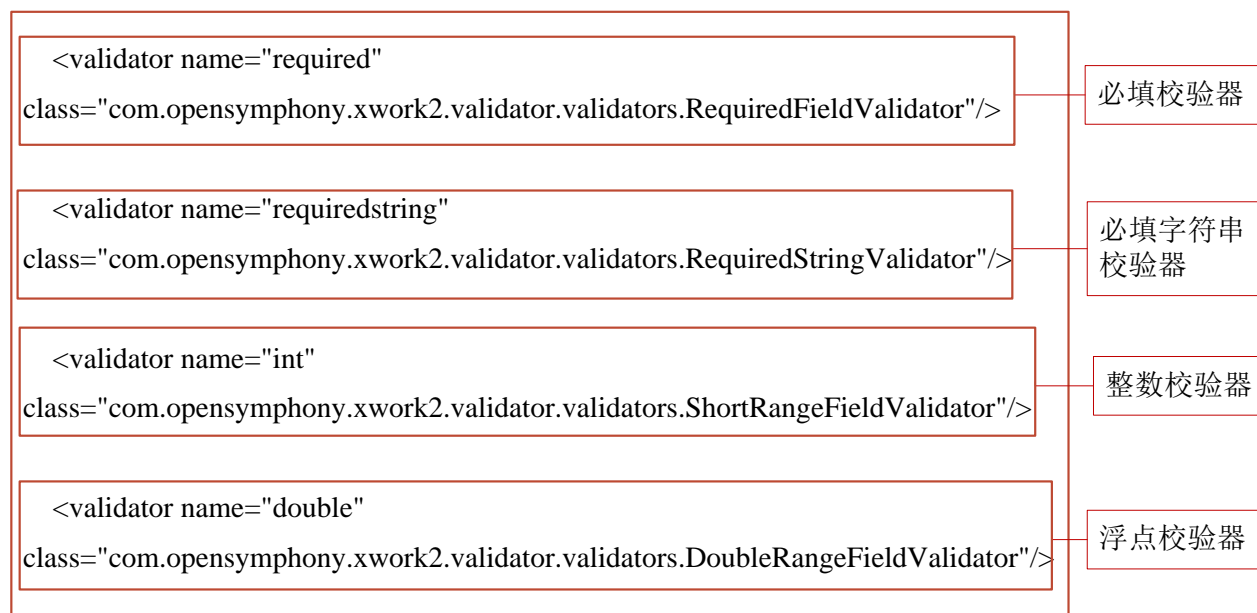
11.3.1 使用validate方法完成输入校验

- * 使用validate方法完成输入校验的这个工作流程可以用图11.27来展示。



11.3.2 Struts 2内置校验框架

- * Struts 2框架提供了一套内置校验框架，可以实现应用中的大部分校验。通过校验框架能够非常简单和快速地完成输入校验。
- * 解压xwork-core-2.3.4.1.jar文件，在xwork-core-2.3.4.1\com\opensymphony\xwork2\validator\validators目录下可以找到default.xml文件，该XML文件定义了Struts 2框架的内建校验器。我们可以截取一部分内容如图11.28所示。



11.3.2 Struts 2内置校验框架

- * Struts 2内建的校验器很多，最常用的有必填校验器、整数校验器、日期校验器和字符串长度校验器等。我们在这里就不一一为大家介绍了，只选取一个必填校验器作为示例来为大家讲解如何使用内建校验器。必填校验器一般用于校验必须输入的属性，要求属性必须有值（非null），其参数fieldName指定校验器的字段名称，如果是字段校验，则不用指定该参数。

11.4 小结

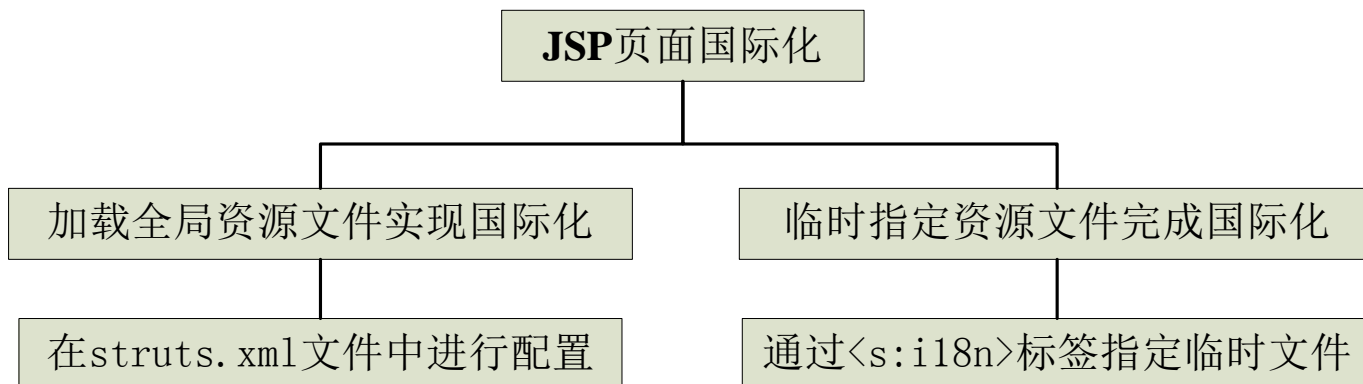
- * 本章主要讲述了Struts 2框架关于类型转化和输入校验方面的知识。类型转化和输入校验是Struts 2框架的重要内容，集中体现了Struts 2框架的优势所在。本章的重点是掌握内建类型的转换器和使用内建校验类型框架对其进行校验。难点内容是理解集合类型属性的类型转换以及使用validate方法完成输入校验的方法。希望读者多加练习，争取掌握。

第12章 国际化和文件上传

- * 随着网络的高速发展，大部分的Web站点已经走出国门，迈向世界了。这是站点的访问者也不再仅仅是本地的浏览者，而是来自于全世界的访客，这时程序国际化成为了Web站点不可或缺的部分。而文件的上传也是Web应用中经常要使用的功能。基于Struts 2的Web应用使国际化开发和文件上传功能都变得非常简单易用，下面我们就来一起学习这两部分知识。

12.1 JSP页面国际化

- * 要实现JSP页面国际化，首先要添加并配置相应的国际化资源文件；然后在JSP页面中通过Text标签指定name参数为相应的Key值，从而实现国际化。JSP页面国际化主要有两种方式，如图12.1所示。



12.1.1 加载全局资源文件实现国际化

- * 一般情况下，可以将国际化信息放到国际化资源文件中，然后在struts.xml文件中配置该文件为全局资源文件。这样就可以在JSP文件中很方便地访问到该资源文件，从而实现国际化。

12.1.2 临时指定资源文件完成国际化

- * 前面示例中通过在struts.xml文件中进行配置全局资源文件，这样的资源文件在每个JSP文件中都可以使用。那如何才能使得资源文件不通过struts.xml配置，同样能被JSP页面调用呢？
- * 我们可以在需要调用国际化资源文件的JSP页面中通过<s:il8n>标签来指定临时资源文件，这样就可以通过指定key值来找到获得指令的国际化信息了。
- * 注意：为了避免全局资源文件的影响，读者可以先把struts.xml文件中的配置全局资源文件的代码删除，这样就可以排除全局资源文件的影响了。

12.1.3 为资源文件传递参数

- * 我们可以在Text标签中添加一个<s:param>标签来传递参数给资源文件。资源文件可以通过占位符的形式来接受参数值。修改前面的资源文件，为资源文件添加占位符。
- * 首先我们修改中文资源文件
jspInterResource_zh_CN.properties，其代码如图12.10所示。

```
welcomeMessage = {0}\uFF0C\u4F60\u597D\uFF01
```

设置占位符

12.1.3 为资源文件传递参数

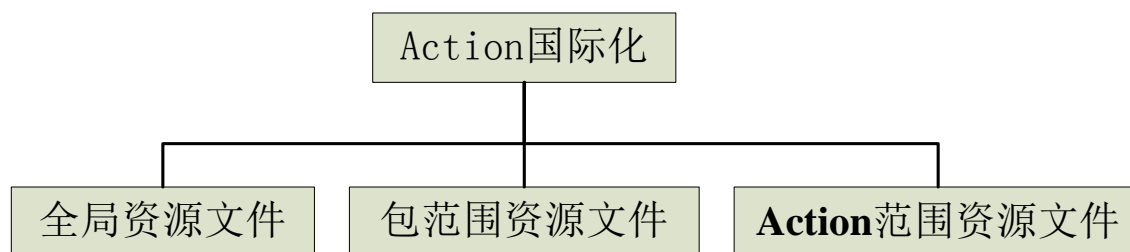
- * 然后再对英文资源文件 `jspInterResource_en_US.properties` 进行修改，具体代码如图12.11所示。

```
welcomeMessage = Hello, {0} \!
```

设置占位符

12.2 Action国际化

- * Struts 2中的Action实现国际化的方式主要有3种，如图12.14所示。



- * 下面我们就来介绍这3种加载方式，并对这3中加载方式的执行顺序加以说明。

12.2.1 加载全局资源文件完成国际化

- * Action可以通过加载全局资源文件完成国际化。全局文件必须存放在项目的WEB-INF/classes目录下。我们首先在src文件夹中完成中文和英文资源文件GlobalInterResource_zh_CN.properties和GlobalInterResource_en_US.properties的添加工作。具体代码如图12.15所示。

GlobalInterResource_zh_CN.properties

```
ActionMessage=\u5168\u5C40\u8303\u56F4-\u6B22\u8FCE\u4FE1\u606F
```

GlobalInterResource_en_US.properties

```
ActionMessage=Global-WelcomeMessage
```

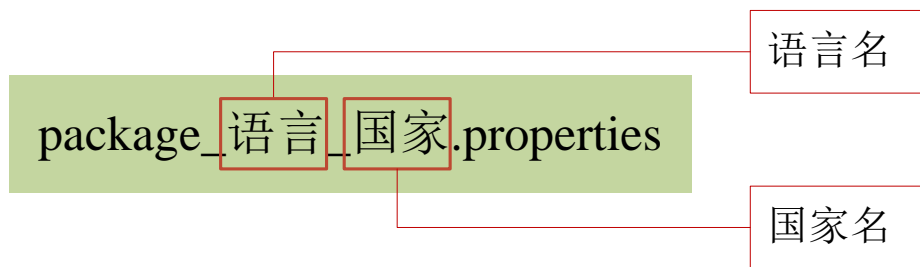
12.2.1 加载全局资源文件完成国际化

- * 然后打开struts.xml配置文件，在struts根节点下添加一个常量节点constant用来配置全局资源文件。具体代码如图12.16所示。

```
<struts>  
<constant name="struts.custom.i18n.resources" value="GlobalInterResource"></constant>  
</struts>
```

12.2.2 加载包范围资源文件完成国际化

- * 下面我们来介绍包范围资源文件。包范围资源文件的好处在于不需要在struts.xml文件中配置。而且不同包下的Action使用不同的包文件，能够很好的将资源文件进行归类。包范围资源文件可以也只能被该包下的Action访问。
- * 包范围资源文件的命名规则如图12.21所示。



12.2.2 加载包范围资源文件完成国际化

- * 然后我们在action包中添加中英文的包范围资源文件。具体代码如图12.22所示。

```
package_zh_CN.properties
```

```
ActionMessage=\u5305\u8303\u56F4-\u6B22\u8FCE\u4FE1\u606F
```

```
package_en_US.properties
```

```
ActionMessage=Package-WelcomeMessage
```

- * 打开IE浏览器，在浏览器地址栏中输入
<http://localhost:8080/ch12/showActionMessage.action>，
分别在中英文页面上显示运行结果。

12.2.3 加载Action范围资源文件 完成国际化

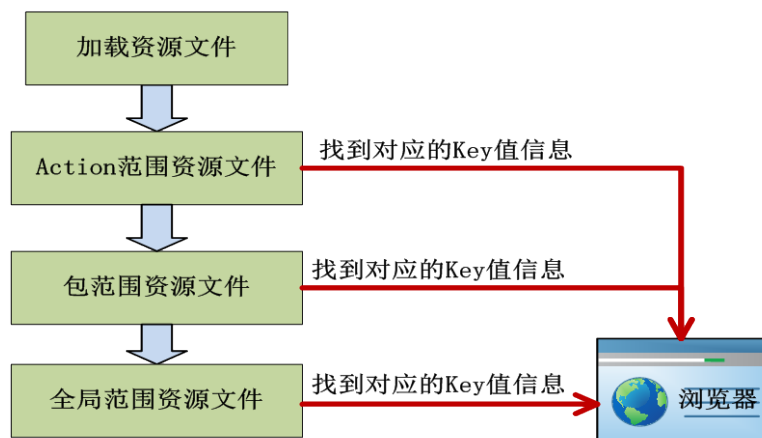
- * 最后我们介绍Action范围资源文件，Action范围资源文件同样不需要在struts.xml文件中配置。Action范围资源文件只能被所对应的Action访问，其他Action无法访问。
- * Action范围资源文件的命名规则如图12.24所示。



- * Action范围资源文件必须和对应的Action类保存在同目录下。我们在action包中添加中英文的Action范围资源文件。
- * 打开IE浏览器，在浏览器地址栏中输入 <http://localhost:8080/ch12/showActionMessage.action>，分别在中英文页面上显示运行结果。

12.2.4 资源文件加载顺序

- * 下面我们对这3种资源文件的加载顺序作以说明，如图12.27所示。



12.3 基于Struts 2完成文件上传

- * Struts 2框架中并没有提供文件上传，而是通过Common-FileUpload框架或COS框架来实现的。Struts 2在原有上传框架的基础上进行了进一步的封装，从而大大简化了文件上传的开发应用。

12.3.1 下载并安装Common-FileUpload框架

- * Common-FileUpload框架是一个非常出色的上传框架，由Apache开源组织的jakarta项目组负责组织维护和更新。为了下载Common-FileUpload框架，我们首先要访问Apache官方站点：<http://www.apache.org/>，在屏幕下方找到“Commons”链接，并在其众多子项目中找到FileUpload和IO两个子项目。

12.3.1 下载并安装Common-FileUpload框架

- * FileUpload和IO两个子项目的下载。
- * 下载完成后，等到两个压缩文件包，分别为commons-fileupload-1.2.2-bin.zip和commons-io-2.4-bin.zip。要安装Common-FileUpload框架，只需将commons-fileupload-1.2.2-bin\bin路径下的commons-fileupload-1.2.2.jar库和commons-io-2.4-bin目录下的commons-io-2.4.jar文件复制到Web应用中的WEB-INF\lib目录中就可以了。

12.3.2 实现文件上传控制器

- * 要实现文件上传，需要修改设置表单的enctype属性。默认情况下，这个值为application/x-www-form-urlencoded，这时只能用来提交普通文本，不能用于文件上传；只有设置为multipart/form-data，才能完整地传递文件数据并完成文件上传操作。

12.3.3 完成文件上传

- * 文件上传控制器创建完成后，需要在struts.xml文件中进行配置，在配置之前首先要填上上传表单页fileUploadPage.jsp和上传结果页fileUploadResultPage.jsp。
- * 上传表单页包含一个文件域和两个按钮，其表单提交到上传文件控制器。
- * 上传结果页，用来显示文件上传结果，包括上传文件的名称以及类型。
- * 然后我们对struts.xml文件进行配置。
- * 配置完成后，就可以开始上传文件了。打开IE浏览器，在浏览器地址栏中输入
<http://localhost:8080/ch12/fileUploadPage.jsp>。

12.3.3 完成文件上传

- * 单击上传按钮完成上传，我们就可以在页面和上传目录中查看文件信息了。
- * 注意：在文件上传之前，要先在WebRoot文件夹下创建好上传目录，而且文件上传后并不会保存到MyEclipse项目中的upload目录下，而是上传到项目的发布目录中，及Tomcat文件夹下的upload目录中。

12.4 多文件上传

- * 文件上传在很多项目中都需要用到，比如博客系统、论坛系统以及邮件系统等。这些系统通常需要上传多个文件，单个文件上传已经不能满足要求了。所以我们就一起来看Struts 2是如何完成多文件上传的。

12.4.1 实现多文件上传控制器

- * 我们在项目中的action文件夹下创建一个多文件上传控制器SomeFileUploadAction.java，该控制器负责封装所有上传文件、文件名和文件类型。

12.4.2 完成多文件上传

- * 同样，在多文件上传控制器创建完成后，需要在struts.xml文件中进行配置，在配置之前首先要填上传表单页SomeFileUploadPage.jsp和上传结果页SomeFileUploadResultPage.jsp。
- * 注意：在上传表单中也可以选择上传一个或两个文件，但是如果一个都不传，就会抛出空指针异常。

12.5 小结

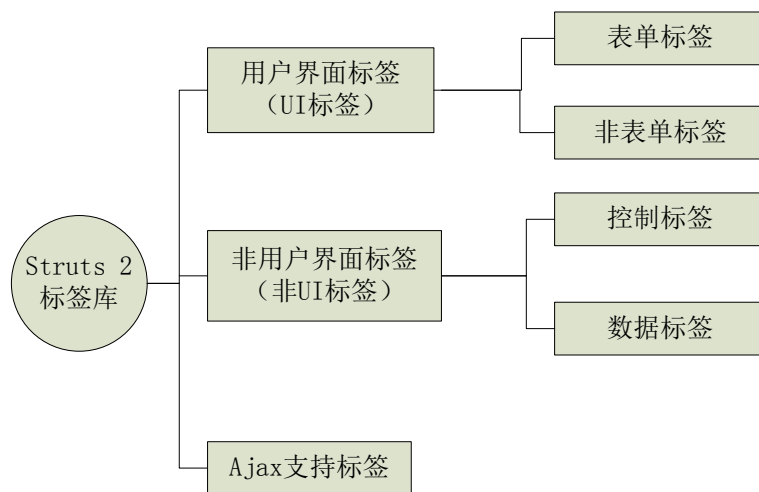
- * 本章主要讲述了Struts 2框架关于国际化和文件上传方面的知识。国际化是Struts 2框架的重要内容，集中体现了Struts 2框架的普遍应用性，而文件上传又是一个非常实用的功能。本章的重点是掌握Action实现国际化的3种方式以及完成文件上传的方法。难点内容是学会并掌握多文件上传的应用。希望读者多加练习，争取掌握。

第13章 Struts 2标签库

- * Struts 2同Struts 1一样，为页面开发提供了大量的标签，但是相比而言Struts 2的标签库更为强大。因为Struts 2不仅整合了Dojo技术，能够生成大量的页面效果，而且它支持OGNL表达式，不再依赖任何表现层技术。借助于Struts 2标签来开发页面，可以使页面更加整洁而且容易维护，同样可以减少代码量以及开发时间。

13.1 Struts 2 标签库概述

- * Struts 2 标签库是一个比较完善且功能强大的标签库。该标签库大大简化了视图页面代码，提高了视图页面的维护效率。Struts 2 并没有严格地对标签进行分类，不过我们可以大体将其分成三大类，如图13.1所示。



13.1 Struts 2 标签库概述

- * 其中，UI 标签主要用于生成 HTML 页面元素；非 UI 标签主要用于数据逻辑输出和数据访问等；Ajax 标签主要用于 Ajax 技术。本章我们主要对非 UI 标签和 UI 标签加以讨论。

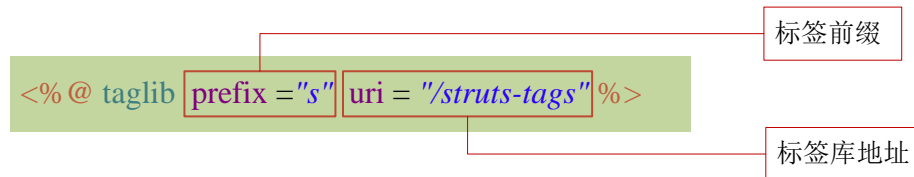
13.1 Struts 2 标签库概述

- * 非UI标签主要用于数据访问和逻辑控制，按其功能又可以分为两类：一类是控制标签，一类是数据标签。表13.1中列出了所有的非UI标签及其功能。

标签分类	标签名	功能
逻辑控制标签	if	条件判断
	elseif	条件判断
	else	条件判断
	append	将集合以追加的方式合并成新集合
	generator	将字符串分割成多个子串
	iterator	迭代数组或者集合
	merge	将集合以交替方式合并成新集合
	sort	对集合中元素进行排序
	subset	获得集合的子集合
数据访问标签	action	在页面中调用Action
	bean	实例化JavaBean
	data	格式化时间和日期
	debug	显示调试信息
	i18n	指定国际化资源文件的文件名
	include	包含JSP等Web资源
	param	传递参数
	property	输出指定值
	push	将指定值放入ValueStack的栈顶
	set	创建一个新变量，并保存在指定范围
	text	输出国际化信息
	url	生成url地址

13.1 Struts 2 标签库概述

- * 要使用Struts 2的标签必须在JSP页面中使用taglib指令来导入Struts 2的标签，引入方式如图13.2所示。



13.1 Struts 2 标签库概述

- * 通过上面的配置就可以在JSP页面中使用Struts 2提供的标签了。使用Struts 2标签的语法格式分为两种，如图13.3所示。

没有标签体:

```
<s:标签名 属性1=属性值1 属性2=属性值2 ...../>
```

有标签体:

```
<s:标签名 属性1=属性值1 属性2=属性值2 .....>  
</s:标签名>
```

13.2 控制标签

- * 控制标签属于非UI标签，它主要用来完成条件逻辑、循环逻辑的控制，也可以用来对集合进行合并、排序操作，还可以对字符串进行分割操作。控制标签一共包含9个标签，我们选取其中常用的6个标签为大家介绍其用法。

13.2.1 if/elseif/else 标签

- * if/elseif/else 标签是3个标签的组合，这3个标签都用于进行条件逻辑控制。其中if标签、elseif标签中提供了一个test属性用来进来判断，该属性返回一个布尔值，通过判断该布尔值来决定是否执行标签体的内容。
- * 注意：if标签可以单独使用，也可以与else标签、elseif标签组合使用。但是else标签和elseif标签不能单独使用，必须和if标签组合使用。

13.2.2 iterator 标签

- * iterator 标签一般用来对集合进行遍历，这里所指的集合包括数组、List、Set 以及 Map 对象。iterator 标签中有 3 个属性，如表 13.3 所示。

属性名	必选	默认值	属性值类型	说明
id	否	无	String	指定引用当前便利元素的ID
status	否	false	Boolean	指定遍历时IteratorStatus实例
value	否	无	String	指定被遍历的集合

- * iterator 标签一般用来遍历 Action 类的集合属性，这里为了为了演示方便，将在 JSP 页面中创建该集合用来模拟从 Action 获得的集合。可以使用 OGNL 表达式中的 “{e1,e2,e3……}” 来生成一个 List 类型集合。

13.2.3 append 标签

- * append 标签用于将多个集合拼接组合成一个新集合。该标签只包含 id 一个属性。该属性用来指定新集合的名字。在 append 标签中可以指定多个 param 子标签。每一个 param 标签指定一个需要被组合的集合。

12.2.4 generator 标签

- * generator 标签用来将指定的字符串按照指定的分隔符分隔成多个子字符串，并将这些子字符串放置到一个集合对象中。转换后的集合对象可以使用 iterator 标签来迭代输出。generator 标签包含 5 个属性，如表 13.4 所示。

属性名	必选	默认值	属性值类型	说明
converter	否	无	org.apache.struts2.util.Converter	用来将集合中的字符串转换成对象
count	否	无	Integer	指定生成集合的元素数量
id	否	无	String	指定级和存储于page范围的变量名
separator	否	无	String	指定分隔符
val	否	无	String	指定被解析的字符串

13.3 数据标签

- * 数据标签主要用来提供数据访问相关的功能，如通过action标签可以显示Action中的属性，通过bean标签允许直接在JSP页面中创建JavaBean示例，使用date标签可格式化日期和时间等。

13.3.1 action标签

- * action标签允许在JSP中直接访问并调用Action，还可以通过executeResult属性选择是否将处理结果包含在当前页面中。action标签包含的属性如表13.5所示。

属性名	必选	默认值	属性值类型	说明
executeResult	否	false	Boolean	指定是否将Action处理结果对应的视图资源包含到JSP页面
flush	否	true	Boolean	指定是否刷新。此操作将写入所有已缓冲的输出字符
id	否	无	String	指定引用该Action的ID
ignoreContextParams	否	false	Boolean	指定是否参数，如为false，将参数传入调用的Action
name	是	无	String	指定调用的Action
namespace	否	当前namespace	String	指定调用Action对应的namespace

13.3.2 bean 标签

- * bean 标签允许直接在JSP页面中创建JavaBean实例。在创建JavaBean对象时，如果需要设置JavaBean的属性，可以在标签体内使用param标签传递参数。bean 标签属性如表13.6所示。

属性名	必选	默认值	属性指属性	说明
id	否	无	String	指定JavaBean实例存储的变量名
name	是	无	String	指定JavaBean对应的完整类名

13.3.3 date标签

- * date标签用来格式化输出指定的时间或者日期。除此之外，date标签还可以输出指定日期到当前时刻的时间差。date标签属性如表13.7所示。

属性名	必选	默认值	属性值类型	说明
format	否	无	String	指定格式化样式
id	否	无	String	指定引用该元素的ID
name	是	无	String	指定引用该元素的名称
nice	否	false	Boolean	指定是否输出指定日期到当前时刻的时间差

13.4 表单标签

- * 表单标签用来向服务器提交用户输入信息，绝大部分表单标签都有相应的HTML标签与其对应，如
`<s:textfield>`标签同`<input type="text".../>`标签，
`<s:password>`标签同`<input type="password".../>`标签。
通过表单标签可以简化表单开发，还可以实现HTML中难以实现的功能，如日期选择器等。

13.4.1 简单表单标签

- * 下面我们来看一组，这些标签都可以在HTML中找到其对应的标签，标签列表如表13.8所示。

标签	HTML对应标签	说明
<s:form>	<form>	表单标签
<s:textfield>	<input type = "text">	单行文本框
<s:textarea>	< textarea >	文本域
<s:submit>	<input type = "submit">	提交按钮
<s:select>	< select >	下拉列表框
<s:reset>	<input type = "reset">	重置按钮
<s:radio>	<input type = "radio">	单选按钮
<s:password>	<input type = "password">	密码输入框
<s:checkbox>	<input type = "checkbox">	复选按钮

13.4.2 combobox 标签

- * combobox 标签用来生成一个单行文本框和下拉列表框的组合，而且这两个元素对应同一个参数。
combobox 标签多用于提示信息或者密码保护问题询问等情况。combobox 标签包括的常用属性如表 13.9 所示。

属性名	必选	默认值	属性值类型	说明
name	否	无	String	指定组合框名称
label	否	无	String	指定组合框前显示文本
list	是	无	String	指定组合框选项集合

13.4.3 datetimepicker 标签

- * datetimepicker 标签用来生成一个文本框和日期、时间选择器的组合。在选择器中选择完某个日期或者时间时，会自动将被选择的日期或者时间输入文本框中。datetimepicker 标签包括的常用属性如表 13.10 所示。

属性名	必选	默认值	属性值类型	说明
displayFormat	否	无	String	指定日期显示格式
displayWeeks	否	6	Integer	指定显示的星期数
endDate	否	2941-10-12	Date	指定最后的日期
formatLength	否	short	String	指定日期格式
label	否	无	String	指定日期选择器前显示文本
name	否	无	String	指定日期选择器名称
startDate	否	1492-10-12	Date	指定最开始日期
type	否	date	String	指定日期选择类型
value	否	无	String	指定默认初始化时间

13.5 小结

- * 本章主要讲述了Struts 2标签库方面的知识。标签库是Struts 2框架的核心内容，集中体现了Struts 2框架的优越性。通过标签库不仅可以减少代码量以及开发时间，而且能够生成大量的页面效果。本章的重点和难点都是能否熟练掌握控制标签、数据标签以及表单标签的用法。运用Struts 2标签库可以给我们的编程工作带来极大的便利，希望读者多加练习，争取掌握。

第14章 Hibernate框架入门

- * Hibernate是目前最流行的持久层框架，专注于数据库操作。使用Hibernate框架能够使开发人员从繁琐的SQL语句和复杂的JDBC中解脱出来。本章将详细介绍什么是ORM以及其优势，然后我们会为大家演示如何为项目中添加Hibernate支持，最后通过一个实际项目介绍开发Hibernate程序的基本步骤和开发技巧。

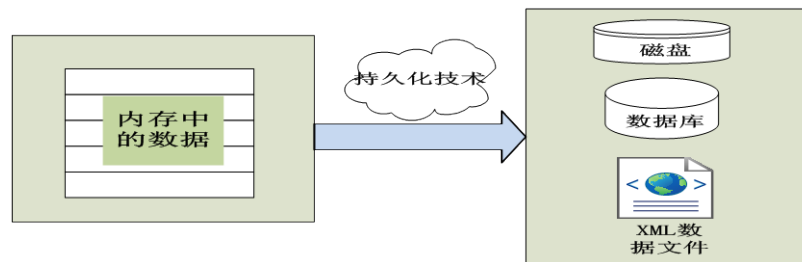
14.1 Hibernate概述

- * Hibernate，如图14.1所示，是一个开放源代码的对象关系映射框架。它对JDBC进行了非常轻量级的对象封装，使得Java程序员可以随心所欲的使用对象编程思维来操纵数据库。Hibernate可以应用在任何使用JDBC的场合，既可以在Java的客户端程序使用，也可以在Servlet/JSP的Web应用中使用。Hibernate的对象关系映射是非常强大并高性能的，其目标是使开发人员从95%的数据持久化工作中解脱出来。



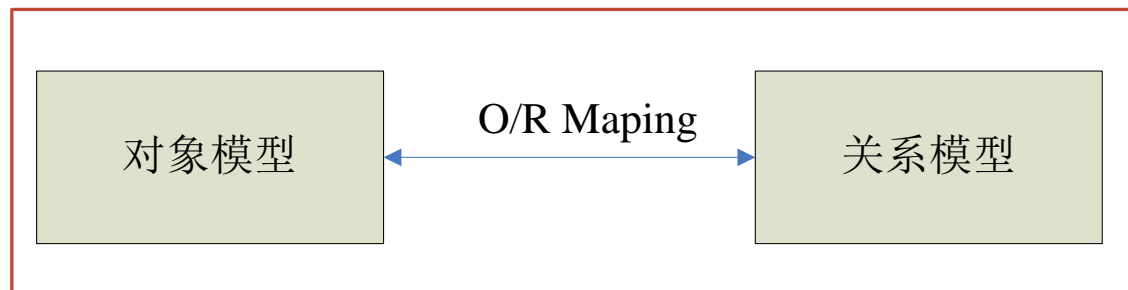
14.1.1 什么是ORM

- * 在了解ORM之前，我们先来了解一下什么持久化技术。持久化技术，就是把数据（如内存中的对象）保存到可永久保存的存储设备中（如磁盘）。持久化的主要应用是将内存中的对象存储在关系型的数据库中，当然也可以存储在磁盘文件中、XML数据文件中等等，如图14.2所示。



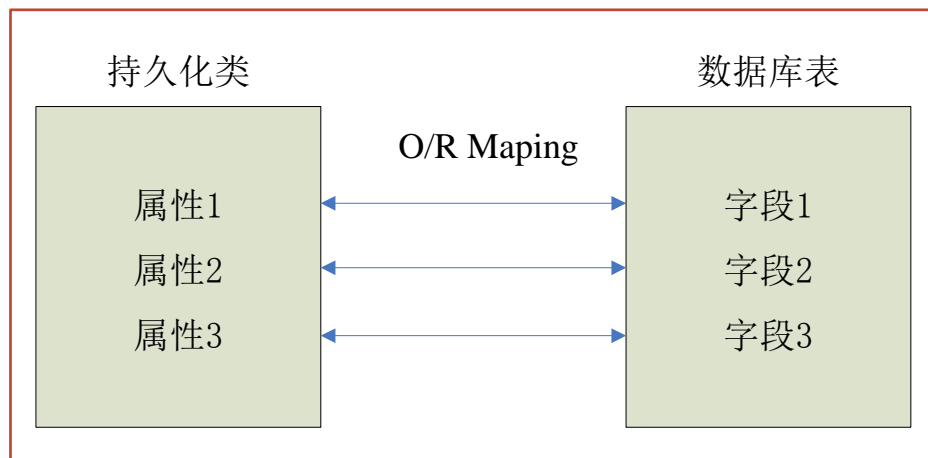
14.1.1 什么是ORM

- * 持久化主要是和数据库打交道的层次，在数据库中对数据的增加、删除、查找和修改都是通过持久化来完成的。
- * ORM (Object/Relational Mapping, 对象/关系映射) 是一种常用的持久化技术。我们所说的对象是指使用的编程语言是面向对象的，则关系则是指使用的数据库是关系型数据库。ORM的映射关系可以用图14.3来表示。



14.1.1 什么是ORM

- * 使用ORM之后就不再需要与复杂的SQL语句打交道了。通过创建一个持久化类来映射数据库的一个数据库表，如图14.4所示。其中持久化的属性则映射到数据库表中的字段。当使用面向对象的方式来操作持久化对象时，ORM框架能自动将这些操作转化成SQL语句，从而完成对数据库的操作。



14.1.2 为什么要使用ORM

- * 现在我们知道了什么是ORM，但是读者可能会有疑问，我已经能够熟练使用JDBC编程了，为什么还要使用ORM呢？使用ORM是整个软件业发展的趋势，下面我们从代码、架构及性能3方面来分析为什么要使用ORM，使用它的好处究竟有哪些。

1. 大大简化了代码

JDBC编程

代码量大
编写繁琐
容易出错

使用ORM

开发人员可以像
操作对象一样来
操作数据库，大
大降低了代码量



2. 将数据库底层透明化

JDBC编程

需了解数据库信息，有哪些表，甚至有哪些字段，对程序员要求高

使用ORM

数据库透明化，开发人员只需要根据业务逻辑需求操作Java对象。不需要了解数据库信息



3. 性能大大优化

JDBC编程

效率低，若在
表中插入1000
条数据要执行
1000次SQL语句

使用ORM

自动向后台数据库
发送SQL请求，多
次请求可以简化成
一次操作。大大降
低通信量



14.1.3 使用Hibernate的优势

- * 目前比较流行的ORM框架主要有Hibernate、iBATIS以及最新的EJB3版本。iBATIS框架并没有真正实现ORM框架，而EJB3是重量级开发框架，不适合轻量级开发。
- * Hibernate框架是一个完整的持久层解决方案，通过Hibernate的支持，可以使用面向对象方式进行各种数据库操作，从而取代传统的JDBC数据库操作。有关Hibernate的优势，我们可以用图14.8来表示。

Hibernate的优势

Hibernate是免费的、开放源代码的

Hibernate是轻量级开发，实现O/R映射简单

Hibernate可扩展性强

Hibernate拥有众多社区支持

14.2 在程序中使用Hibernate

- * 在应用程序中使用Hibernate框架非常简单。只要在CLASSPATH环境变量中指定Hibernate框架的jar包，就可以在程序中像使用其他的jar包一样使用Hibernate。但要想使用Hibernate框架，需要进行一些配置。如果系统比较大的话，将会产生非常大的工作量。因此，要想更好的使用Hibernate，就需要一个支持Hibernate的IDE，如MyEclipse。由于我们在程序中经常要使用数据库，我们先来安装一个MySQL数据库。

14.2.1 安装MySQL数据库

- * MySQL数据库是一个小型的关系型数据库。它体积小、速度快，而且是免费、开源的。对于一般的个人使用者来说，MySQL数据库提供的功能和性能已经绰绰有余，尤其适合初学者学习使用。本书与数据库相关的实例都是基于MySQL数据库的。
- * 为了安全起见，我们建议大家从官网下载MySQL，MySQL数据库下载的官方网址是<http://dev.mysql.com/downloads/>。

14.2.2 MyEclipse对Hibernate的支持

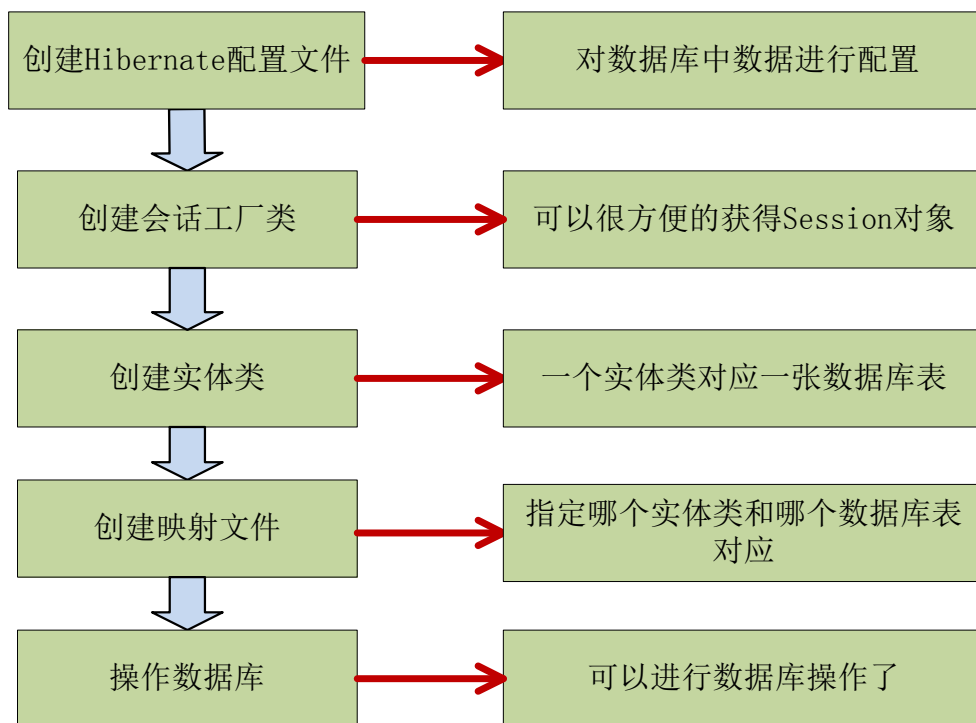
- * 对MyEclipse建立的工程在默认情况下是不支持Hibernate的，需要按照下面的步骤操作操作，才能为工程添加Hibernate支持。
- * 由于读者所使用的MyEclipse可能不支持MySQL数据库，我们先将MySQL数据库应用添加到MyEclipse开发工具中去。我们启动MyEclipse，选择“Windows”|“Open Perspective”|“MyEclipse Database Explorer”命令。在弹出的DB Browser区域内，右击选择菜单项“New”，建立一个新连接。

14.3 第一个Hibernate程序

- * 在完成了MyEclipse对Hibernate的支持配置之后，本节通过一个Java应用程序来演示如何使用Hibernate完成持久化操作。通过该项目读者可以了解开发Hibernate程序的基本步骤，感受无SQL操作数据库的魅力所在。

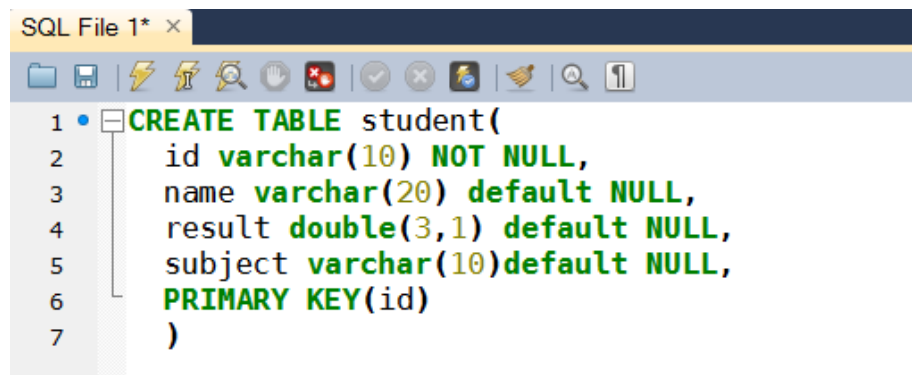
14.3.1 开发Hibernate程序的基本步骤

- * 在开始实际项目开发之前，我们首先来了解一下开发Hibernate程序的基本步骤，其具体内容及相关说明可以用图14.18来表示。



14.3.2 创建数据库

- * 我们在MySQL数据库中创建一个数据库表student，用来保存学生信息。student表一共包含4个字段，分别用来保存学生的学号、姓名、科目和成绩。创建student表的SQL语句如图14.19所示。



```
SQL File 1* x
1 • CREATE TABLE student(
2   id varchar(10) NOT NULL,
3   name varchar(20) default NULL,
4   result double(3,1) default NULL,
5   subject varchar(10) default NULL,
6   PRIMARY KEY(id)
7 )
```

14.3.3 创建Hibernate配置文件

- * Hibernate从其配置文件中读取和数据库有关的信息。前面我们通过MyEclipse自动创建了一个XML格式的配置文件，文件名为hibernate.cfg.xml。该文件中配置了数据库连接URL、数据库连接驱动、数据库用户名以及用户密码。这里还配置一个属性dialect，用来指定数据库产品类型。如果需要调试Hibernate，可以将show_sql属性设为true，这样Hibernate在进行持久化时会将相应的SQL语句输出到控制台。

14.3.4 创建会话工厂类

- * 在使用Hibernate操作数据类型时，需要得到一个Session对象。通过调用Session对象的方法，才能完成数据库操作，如添加记录、删除记录等。这个HibernateSessionFactory.java也是由系统自动生成的。它即存放在我们在14.2.2小节中自己创建的包中，我们在这里就不为大家展示了。

14.3.5 创建实体类

- * 实体类用来映射数据库中的一张数据库表，实体类中的属性与数据库表中的字段相对应。持久化类是一个POJO类（简单的Java对象），不用继承和实现任何类或接口。

14.3.6 创建对象关系映射文件

- * 关系映射文件用来映射持久化类和数据库表，从而将持久化类的属性和数据库表中的字段联系起来。其中id元素用来定义主键标识，property元素用来定义其他属性。映射文件的文件名一般为持久化类名加上“.hbm.xml”，文件保存在持久化类同目录下。

14.3.7 完成插入数据

- * 如果需要使用Hibernate来插入数据，需要调用Session对象的save方法。save方法接受一个Object类型的参数，该参数为一个封装了所有插入数据的对象。

14.3.8 查询学生列表

- * 我们可以按照同样的方法往数据库中多增加几条数据信息。然后我们就可以实现按条件对数据库表中的信息进行查询了。

14.4 小结

- * 本章是Hibernate的入门章节，首先我们为大家介绍什么是ORM，以及为什么要使用Hibernate，然后介绍了MySQL数据库的安装过程以及如何在MyEclipse开发工具中使用Hibernate，最后又通过一个完整的实例，向大家展示了运用Hibernate对数据库进行操作的过程。本章的重点和难点都是理解并争取掌握Hibernate的开发过程，力争为后面章节的学习打下坚实的基础。

第15章 Hibernate配置和会话

- * Hibernate的配置包括两个重要部分，一个是Hibernate的配置文件hibernate.cfg.xml，一个是实体类的映射文件。本章我们将对这两个文件的配置进行详细介绍以及如何使用Annotations配置映射，最后还将介绍Hibernate的3种对象状态及Session的各种方法及应用。

15.1 传统方式配置Hibernate

- * Hibernate的配置主要有两种方式：传统的配置方法以及使用Annotations进行配置的方法。首先我们先来学习传统的配置方式。

15.1.1 配置Hibernate

- * 在使用Hibernate进行持久化之前，必须对Hibernate进行一系列配置，如数据库连接URL、数据库用户名和密码以及映射文件路径等。对于Hibernate的配置，最常用的就是采用XML格式的方法进行配置。Hibernate默认的配置文件为hibernate.cfg.xml，其包含的配置属性如表15.1所示。

属性名	描述
hibernate.dialect	Hibernate方言所对应的类名
hibernate.show_sql	设置是否在控制台输出SQL语句
hibernate.connection.url	设置数据库连接URL
hibernate.connection.username	设置数据库用户名
hibernate.connection.password	设置数据库密码
hibernate.connection.driver_class	设置数据库连接驱动类
hibernate.default_schema	生成SQL时，schema/tablespace的全限定名

15.1.1 配置Hibernate

- * 注意：使用XML文件配置Hibernate时，可以将属性名简写，如将hibernate.show_sql直接写为show_sql。

15.1.2 配置映射文件

- * 映射文件是持久化操作中的一个重点，它是数据库表和实体类之间的连接枢纽。通过映射文件，Hibernate就能知道实体类和那个数据库表相对应。映射文件也是采用XML文档规范，这样设计可以使其非常易读，而且容易修改。下面我们就对其包含的各种元素分别做一简要介绍。

1. <hibernate-mapping>元素

- * 映射文件的根节点为<hibernate-mapping>，该节点包含一系列的可选属性，如schema和catalog属性。schema属性指定了数据库表所在的schema名称。

属性名	必选	默认值	描述
schema	否	无	指定数据库schema的名称
catalog	否	无	指定数据库catalog的名称
default-cascade	否	none	指定默认的级联风格
default-access	否	property	指定访问所有属性的策略
default-lazy	否	true	指定默认加载风格
auto-import	否	true	指定是否可以查询非全限定的类名
package	否	无	指定包前缀，若没有指定全限定的类名，将使用这个作为包名

- * 注意：一个映射文件中只允许有一个<hibernate-mapping>元素。

2. <class>元素

- * <class>元素用来配置一个实体类与一个数据库表的关联。其中name属性用来指定实体类的名称，table属性用来指定数据库表的名称。<class>元素的常用属性如表15.3所示。

属性名	必选	默认值	描述
name	否	无	指定完全路径类名
table	否	无	指定数据库表名
mutable	否	true	指定类的实例是否可变
proxy	否	无	指定代理类接口，为延迟加载提供支持
lazy	否	true	指定是否使用延迟加载
dynamic-update	否	false	指定生成Update SQL时是否仅包含发生变动的字段
dynamic-insert	否	false	指定生成Insert SQL时，是否仅包含非空字段

- * 注意：<hibernate-mapping>元素下可以由多个<class>元素，但是一般推荐值添加一个。即一个实体类对应一个映射文件。

3. <id>元素

- * 每一个实体类中都包含一个唯一的标识，<id>元素能够定义该属性和数据库表中的主键字段的映射。<id>元素包括的常用属性如表15.4所示。

属性名	必选	默认值	描述
name	否	无	指定标识属性的名称，如果不指定，表示这个类没有标识属性
type	否	无	指定标识属性的Hibernate类型
column	否	无	指定数据库表中主键字段的名称
unsaved-value	否	无	指定该实例是刚创建的，尚未进行保存
access	否	property	指定Hibernate用来访问属性值的策略

4. <property>元素

- * 实体类的标识和数据库表的主键映射完成后，还需要为实体类的其他属性和数据库的其他字段进行映射，这个时候就需要使用到<property>元素。
<property>元素的常用属性如表15.5所示。

属性名	必选	默认值	描述
name	否	无	指定标识属性的名称，如果不指定，表示这个类没有标识属性
type	否	无	指定标识属性的Hibernate类型
column	否	无	指定数据库表中主键字段的名称
access	否	property	指定Hibernate用来访问属性值的策略
not-null	否	true	指定属性是否允许为空
generated	否	never	指定属性值是否由数据库生成

15.2 使用Annotations配置映射

- * 在JDK 5.0之后出现了一种新的注释技术Annotations，而Hibernate也在其3.0之后的版本中添加了对Annotations的支持。通过在实体类中添加Annotations注释，可以达到替代映射文件的效果。

15.2.1 使用@Entity注释实体类

- * @Entity注释用来将一个普通的JavaBean标注为实体类。@Entity注释由一个可选的name属性，用来设置属性名。并不是所有的JavaBean都能被标注为实体类，必须要满足如图15.7所示的3个条件。

标注为实体类的3个条件	JavaBean类的访问权限只能是public
	JavaBean类不能是抽象类
	JavaBean中必须有一个访问权限为public的无参的构造方法

- * 注意：使用@Entity注释实体类，一定要使用import语句引入javax.persistence.Entity类，该类为@Entity注释依赖类。

15.2.2 使用@Table注释实体类

- * @Table注释用来对实体类进行进一步注释，用来配置实体类到数据库表映射的更详细的信息。@Table注释包含的属性信息如表15.6所示。

属性名	属性描述
catalog	用来设置数据库名
name	用来设置数据库表名
schema	用来设置数据库表的所有者名称
uniqueConstraints	用来设置数据库表的约束

- * 注意：使用@Table注释实体类，一定要使用import语句引入javax.persistence.Table类，该类为@Table注释依赖类。

15.2.3 使用@Id注释实体类标识

- * @Id注释用来对实体类的标识进行配置。一个实体类一般只有一个标识，所以一个实体类中只出现一个@Id注释。
- * 注意：使用@Id注释实体类标识，一定要使用import语句引入javax.persistence.Id类，该类为@Id注释依赖类。

15.2.4 使用@GeneratedValue注释覆盖标识的默认访问策略

- * 使用@Id注释实体类标识时将采用Hibernate的默认访问策略，这时可以使用@GeneratedValue注释覆盖标识的默认访问策略。@GeneratedValue注释包括两个属性，即使用generator属性指定标识生成器名，使用strategy属性指定标识生成策略。strategy属性的属性值为一个枚举类型，其中包含了4个枚举值，如表15.7所示。

枚举值	描述说明
javax.persistence.GeneratedValue.AUTO	strategy属性默认值，表示自动确定表示的类型
javax.persistence.GeneratedValue.IDENTITY	用来表示由数据库自动设置标识的值，如自动递增字段
javax.persistence.GeneratedValue.SEQUENTITY	用来表示标识为SEQUENTITY类型
javax.persistence.GeneratedValue.TABLE	用来保证另一个使用该标识的表记录的唯一性

15.2.5 使用@GenericGenerator注释生成标识生成器

- * 前面我们介绍了如何使用@GenerateValue注释的strategy属性来指定生成策略，但是这些生成策略明显不能满足。这时可以使用@GenericGenerator注释产生标识生成器，然后通过@GenerateValue注释的generator属性来制定生成器的name属性，这样就可以采用指定的生成器生成标识。
- * @GenericGenerator注释包含3个属性，其属性说明如表15.8所示。

属性名	属性描述
name	用来设置标识生成器名
parameters	用来设置标识生成器所需的参数
strategy	用来设置Hibernate内置的生成策略

15.2.6 使用@Column注释实体类非标识属性

- * 一个实体类除了有标识，一般还会有许多其他属性，这时可以使用@Column注释这些属性。@Column注释最常用的属性为name属性，该属性用来设置数据库表中的字段名。

15.2.7 自定义

AnnotationSessionFactory 类来获得 Session 对象

- * 前面我们介绍了如何使用 Annotation 注释来完成实体类到数据库表的映射，这时还有一点需要特别注意。在以前获得 SessionFactory 对象是通过调用 Configuration 对象来实现的，但是这种方式不支持 Annotations 注释映射。要支持 Annotations 就必须使用 AnnotationConfiguration 类了。

15.2.8 测试Annotations注释是否成功完成映射

- * 通过AnnotationSessionFactory可以加载Annotations注释方式的映射。通过调用Session对象的各种方法就可以完成各类数据库操作，如查询记录、添加记录等。在创建测试类之前，首先同样需要在Hibernate的配置文件hibernate.cfg.xml中添加映射信息。同映射文件配置映射不同，这里需要指定class属性为需要映射的实体类。具体配置方法如图15.15所示。

```
<mapping class="po.Employee" />
```

15.2.8 测试Annotations注释是否成功完成映射

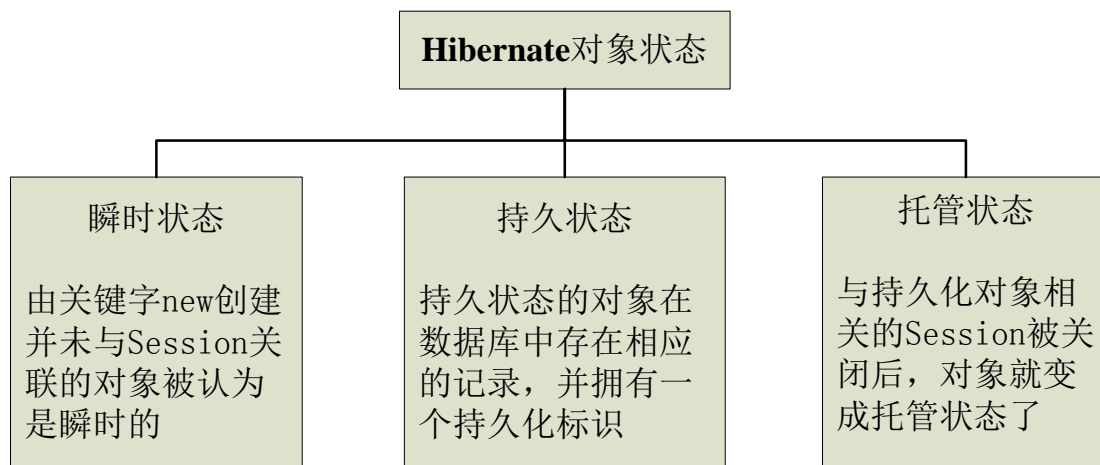
- * 完成如上步骤后，映射信息就已经全部完成了，下面我们就来创建一个测试类来测试使用Annotations注释是否能成功完成映射。
- * 我们首先打开MySQL数据库并在名为mysqltest的数据库中建立一个tb_employee表，并在表中按照前面所述的属性建立一条数据信息。

15.3 会话 (Session) 的应用

- * Hibernate提供了一个会话类Session，可以通过SessionFactory获得Session实例对象。通过调用Session对象的个方法即可完成数据库操作，如通过save方法来插入记录，通过load方法按标识取出记录，通过delete方法删除记录等。

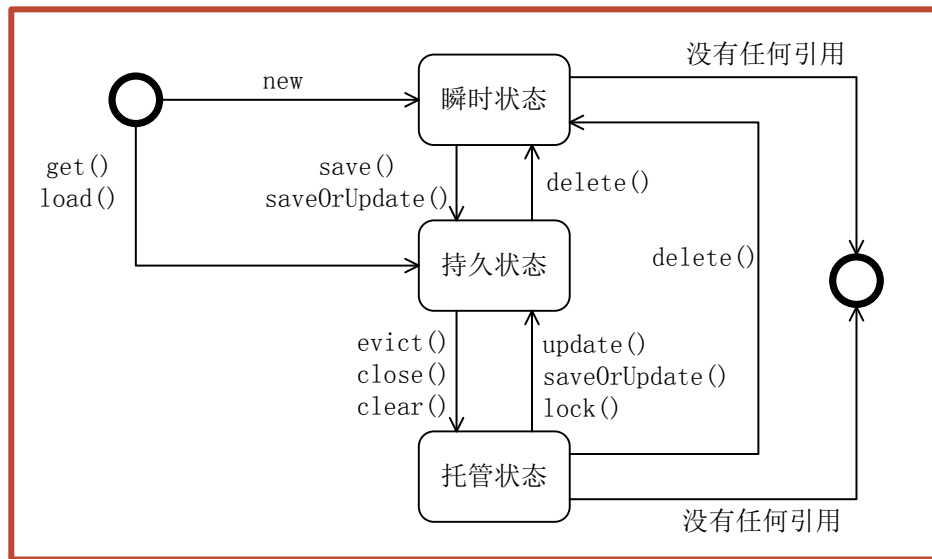
15.3.1 Hibernate对象状态

- * 一个实体类的实例可能处于3种不同的状态中的一种。这3种状态分别为瞬时状态、持久状态和托管状态。下面我们就来看这3种状态的详细说明，如图15.16所示。



15.3.1 Hibernate对象状态

- * 我们总结这3中台，再结合个状态的转换方法就可以得到其状态转换图，如图15.17所示。



15.3.2 使用save方法持久化对象

- * 使用new关键字创建的对象并没有保存到数据库中，这时的对象为瞬时状态。通过Session对象的save方法能够将其转换成持久状态，并同时在数据库表中添加相应记录。save方法有两种重载方式，如图15.18所示。

```
public Serializable save(Object object) throws HibernateException;  
public Serializable save(String entityName, Object object) throws HibernateException;
```

设置实体类的实例名

设置实体类的类名

15.3.3 使用load方法装载对象

- * 如果知道某个对象的持久化标识，就可以使用Session对象的load方法从数据库中装载数据，使用load方法装载的对象是持久状态的。

15.3.4 使用refresh方法刷新对象

- * 使用refresh方法能够根据数据库中的数据来刷新持久对象中的属性值。refresh方法有两种重载方法，如图15.23所示。

设置需要刷新的持久对象

```
public void refresh(Object object) throws HibernateException;  
public void refresh(Object object, LockMode lockMode) throws HibernateException;
```

15.3.5 使用delete方法删除对象

- * 可以使用Session对象的delete方法来删除数据库中的记录。delete方法有两个重载方式，如图15.26所示。

```
public void delete(Object object) throws HibernateException;  
public void delete(String entityName, Object object) throws HibernateException;
```

指定需要删除的持久对象

设置实体类的类名

15.4 小结

- * 本章主要讲述了Hibernate的配置和会话方面的知识，熟练掌握Hibernate的配置和会话是使用Hibernate操纵数据库的基础，也是学习高级Hibernate技术的基础。本章的重点内容是掌握使用Annotations配置映射的方法，难点内容是理解会话（Session）的应用的具体方法。希望读者在学习时多加练习，争取掌握。

第16章 Spring框架入门

- * SSH框架是目前最为流行的软件开发技术，它是由3种技术组成的，除了我们前面讲解的Struts和Hibernate外，还包括从本章开始讲解的Spring，如图16.1所示。Spring是一种非常完善的开源的框架，通过它可以大大降低企业应用程序的复杂性。我们在开发中通常使用Spring开发业务逻辑层。

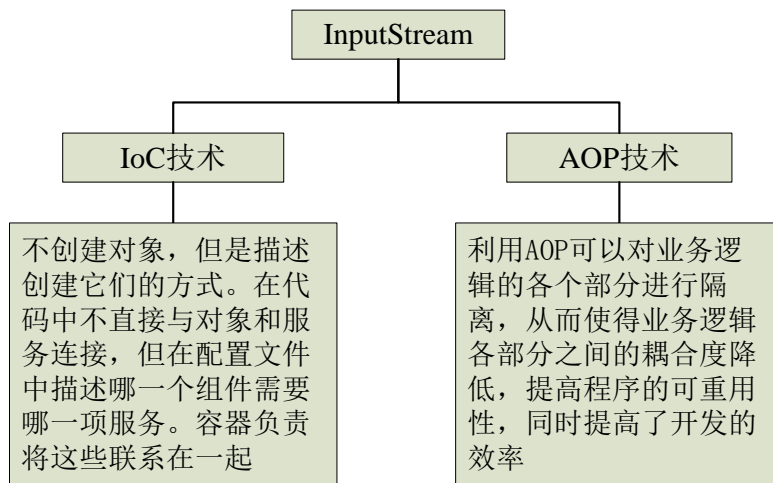


16.1 Spring概述

- * 如果读者在学习Spring之前，学习过EJB技术，就知道开发企业级项目是一件非常复杂的工程。随着Spring的出现，会大大降低J2EE企业级开发的复杂度。作为一种开源技术，Spring几乎替代了EJB技术。并且Spring不仅仅是替代品，其技术范围比EJB更广、更实用。

16.1.1 Spring技术介绍

- * Spring是一种非常完整的技术，即使只使用Spring技术也能实现项目的开发。但是在实际开发中我们只是让Spring做业务逻辑层，因为Spring的业务处理能力是非常强大的。简单来说，Spring就是一个轻量级的控制反转（IoC）和面向切面（AOP）的容器框架，如图16.2所示。



16.1.2 为什么使用Spring

- * 在没有使用Spring之前，如果在业务逻辑层中访问数据访问层，需要在业务逻辑层中创建数据库访问层的对象，然后使用该对象调用DAO方法。
- * 使用这种方式访问数据访问层，当数据访问层程序发生改动时，还需要改动业务访问层的程序，加大了程序员的工作量。
- * 当Spring出现以后，这种问题就得到了解决。业务逻辑层和数据访问层之间是注入的关系，在业务逻辑层中并不需要创建数据访问层的对象，

16.2 Spring开发环境的搭建

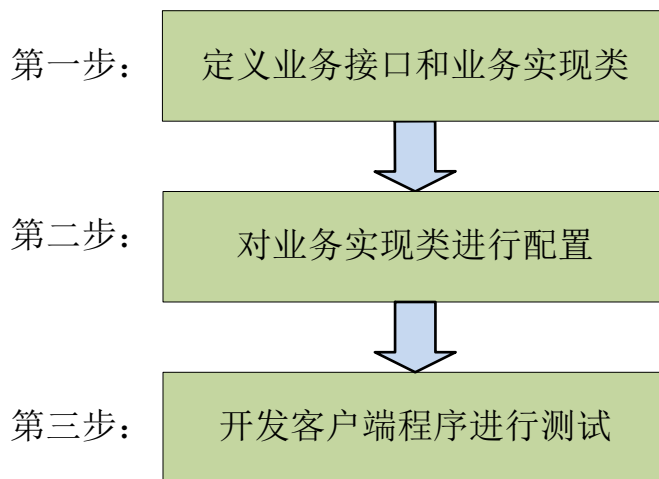
- * 在MyEclipse中集成了Spring项目开发，通过它可以非常容易的搭建Spring开发环境。其搭建步骤大致如下：
- * （1）创建一个Java项目ch16，选中该项目单击右键，在弹出的菜单栏选择“MyEclipse”|“Add Spring Capabilities”命令，然后选择版本号及所要添加的包。我们可以选择导入所有的jar包。
- * （2）单击Next按钮，进入Spring设置页面。在页面中，可以设置Spring的配置文件名和保存目录。默认情况下，Spring配置文件名为applicationContext.xml，建议读者不要修改这个文件名。然后就可以单击Finish按钮完成配置。

16.3 开发Spring的HelloWorld程序

- * 在前面的学习中，我们已经对Spring有了一个初步的了解，本节我们将通过一个非常简单的HelloWorld程序为大家介绍如何使用Spring开发环境进行程序的开发。Spring有两个非常重要的应用，那就是IoC控制反转和AOP面向切面编程，我们这里先以IoC技术为代表进行讲解。

16.3.1 开发Spring程序的步骤

- * Spring开发是有严格步骤的，无论项目简单和复杂，都要按照这个步骤进行操作。Spring程序的开发步骤如图16.7所示。



16.3.2 编写业务接口

- * 我们首先来开发业务接口，在该业务接口中定义了SayHello方法。通过该方法创建一个接收传递信息，然后返回问候语句的功能。

16.3.3 编写业务实现类

- * 开发完业务接口后，就继续来编写业务实现类。业务实现类要实现业务接口，从而实现业务接口中的抽象方法。

16.3.5 编写客户端进行测试

- * 到目前为止，Spring的程序已经开发完毕，本节就通过一个客户端程序来对Spring的程序进行测试。通过该客户端程序调用业务实现类中的业务方法。
- * 学习到这里，读者可能并没有感受到Spring的开发优势，这是因为我们还没有进行实际开发。在Web项目开发中，最重要的一点就是进行分层开发，而Spring就是起到这个作用。当我们的程序需要该懂事，只需要改动Spring的配置文件，这在以前的方式中是不可能做到的。

16.4 小结

- * 本章是Spring的入门章节，首先我们为大家介绍Spring技术，以及为什么要使用Spring，然后介绍了Spring开发环境的搭建，最后又通过一个完整的实例，向大家展示了开发Spring程序的步骤。本章的重点和难点都是理解并争取掌握Spring的开发过程，力争为后面章节的学习打下坚实的基础。

第17章 控制反转

- * 控制反转是Spring的核心技术之一。英文名称为“Inversion of Control”，所以一般将其称为IoC。Spring的很多功能都是基于控制反转技术的。上一章中的HelloWorld程序就是基于控制反转技术的。本章我们将继续学习这一核心技术。

17.1 IoC容器

- * 容器的字面意思可以理解为用于存放其他事物的物品。在Spring中，IoC容器除了具有该含义以外，它还可以对其中的事物进行管理。在Spring中，IoC容器中的事物被称为Bean，它是由IoC容器初始化、装配和管理的对象。

17.1.1 Bean工厂接口

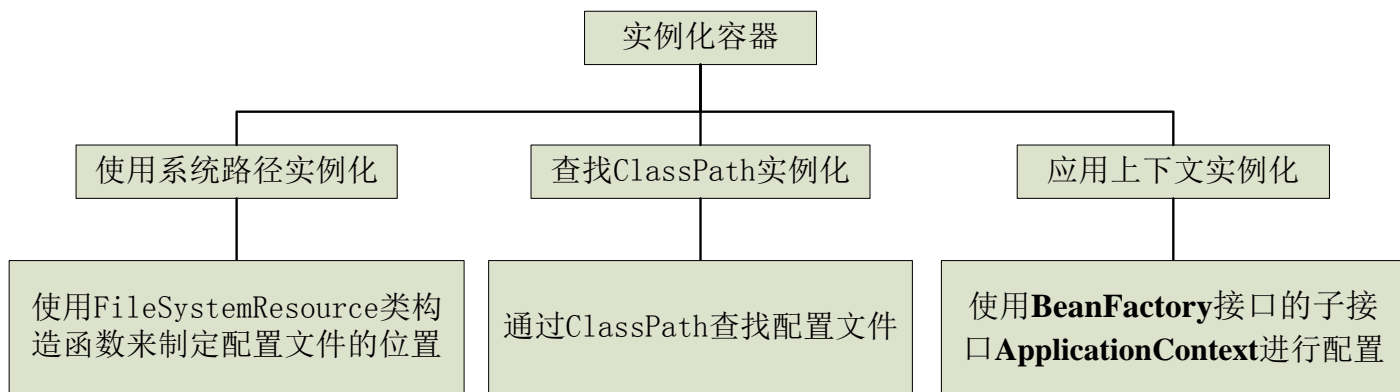
- * Bean工厂接口的全称接口名称为“org.springframework.beans.factory.BeanFactory”。在Spring程序中，BeanFactory实际上是IoC的代表者，通过使用该接口对其中的Bean进行管理。在Spring中，定义了多种Bean工厂接口的实现类，其中最常用的就是XmlBeanFactory，使用该类可以以XML文件的形式来描述对象与对象之间的关系。Bean工厂接口的引入方式如图17.1所示。

```
import org.springframework.beans.factory.BeanFactory;
```

引入**Bean**工厂接口

17.1.2 实例化容器

- * 要想使用IoC容器，就需要对容器进行实例化操作，通过实例化后的容器对象就可以访问其中的Bean对象。实例化容器有多种方式，如图17.2所示。



17.1.3 多配置文件的使用

- * 当开发的项目较大时，使用一个Spring配置文件是比较复杂的，其中可能具有几千行的配置信息，在其中查找某一个Bean对象的配置是非常困难的。所以在Spring中可以定义多个配置文件，将一类配置放在一个文件中。

17.1.4 使用容器实例化Bean

- * 前面我们我们已经见过如何配置Bean。配置Bean是通过使用<bean>标记对其进行配置的，其中class属性指定Bean的完整路径。每一个Bean中都有一个或者多个id属性，id属性是Bean的标识符，在当前IoC容器中必须是唯一的。
- * 对容器实例化后，使用容器对象调用getBean方法，就可以实例化指定的Bean了，具体示例如图17.5所示。

```
HelloService helloService=(HelloService)factory.getBean("hello");
```


17.2 依赖注入

- * 在开发HelloWorld程序时，我们只是将业务实现类配置到Spring配置文件中。但是在实际开发中并不是这样的，除了业务实现类外，还有DAO实现类和控制层类，在项目中它们要结合使用，这时候就要使用到Spring中的依赖注入技术。依赖注入有两种常用的方式，分别是Setter方法注入和构造函数注入。

17.2.1 Setter方法注入

- * 使用Setter方法进行注入是依赖注入的一种方式。在使用Bean中定义需要注入属性的Setter方法，然后在Spring配置文件中给出该属性的值，最后在客户端程序中就可以访问这些属性值。我们先来看一个完整的程序。

17.2.2 构造函数注入

- * 虽然在Spring中提倡使用Setter方法的方式进行注入，但是使用构造函数同样是依赖注入非常重要的方式之一。有些程序员更倾向于使用构造函数的方式进行注入，是因为使用这种方式可以将所有的属性一次性注入。我们也来看一个具体的实例，来看如何实现构造函数的注入。

17.2.3 注入其他Bean

- * 我们还可以将Bean中指定属性的值设置为对容器中的另外一个bean的引用的方式，即可以在一个Bean中注入其他Bean。该操作是通过<ref>标记对完成，将其放在<constructor-arg>标记对或者<property>标记对中。

17.2.4 注入集合

- * 在Java中，集合包括List、Set和Map3种。Spring除了能够对这3种集合进行注入操作外，还包括Properties文件。在Spring配置文件中，使用<list/>、<set/>、<map/>和<props/>标记来对应集合。

17.3 Bean作用域

- * 在Spring配置文件中创建Bean时，还要设置Bean的作用域。Bean的作用域就是指一个Bean对象的使用方式和使用范围。Spring中的这种设计使定义Bean的作用域不用在类程序中进行，而是在配置文件中进行，大大提高了灵活性。在Spring中，支持的5种内置Bean作用域如表17.1所示。

作用域名称	作用域说明
singleton	Bean默认作用域，每个IoC容器中，一个Bean定义对应一个实例对象
prototype	每个IoC容器中，一个Bean定义对应多个实例对象
request	在一次HTTP请求中，一个Bean定义对应一个实例对象
session	在一个HTTP Session中，一个Bean定义对应一个实例对象
global session	在一个全局的HTTP Session中，一个Bean定义对应一个实例对象

- * 其中request、session和global session的作用域只在Web的项目中有效。本节将对这几种作用域进行详细讲解。

17.3.1 singleton作用域

- * singleton作用域是Bean的默认作用域。当一个Bean的作用域为singleton时，在Spring IoC容器中只会存在一个共享的Bean实例。也就是说，所有对Bean的请求，只要id与该Bean定义相匹配，就会返回Bean的同一实例。singleton作用域的定义方式如图17.20所示。

```
<bean id="teacher" class="service.teacher" scope="singleton" />  
<bean id="teacher" class="service.teacher" singleton="true" />
```

17.3.2 prototype作用域

- * prototype作用域是和singleton作用域相对的。当一个Bean设置为prototype作用域时，则该Bean每被请求或者引用一次，都会创建一个Bean实例对象。prototype作用域的定义方式如图17.21所示。

```
<bean id="teacher" class="service.teacher" scope="prototype" />  
<bean id="teacher" class="service.teacher" singleton="false" />
```


17.3.3 request作用域

- * request、session和global session的作用域和前面两种作用域有很大不同，它们只能应用在Web的项目中。Web中容器实例化和前面学过的实例化也是有所不同的，它要使用XmlWebApplicationContext来查找Spring配置文件。而且它们在使用Bean时，要对web.xml文件进行如图17.22所示的配置。

```
<web-app>
  <listener>
    <listener-class>org.springframework.web.context.request.RequestContextListener
  </listener-class>
  </listener>
</web-app>
```

17.3.3 request作用域

- * 在其中配置了监听器，使用RequestContextListener监听器可以将HTTP request对象绑定到位该请求提供服务的Thread。
- * 在Web中，每次HTTP请求就对应一个request对象。将Bean的作用域设定为request，就表示该Bean只在当前请求中有效。request作用域的Bean的配置如图17.23所示。

```
<bean id="UserDAO" class="dao.UserDAOImpl" scope="request" />
```

17.3.4 Session作用域

- * 在Web中，每次HTTP会话就对应一个Session对象。将Bean的作用域设定为session，就表示该Bean只在当前会话中有效。当一次会话结束时，session作用域的Bean的配置如图17.24所示。

```
<bean id="UserDAO" class="dao.UserDAOImpl" scope="session" />
```

17.3.5 global session作用域

- * global session作用域和Session作用域是非常相似的，不过它在开发中是非常少用的。在global session作用域的bean中被限定在全局portlet Session的生命范围内。global session作用域的Bean配置如图17.25所示。

```
<bean id="UserDAO" class="dao.UserDAOImpl" scope="global session" />
```

17.4 小结

- * 本章讲解了Spring的第一个重要部分控制反转，首先我们为大家介绍了IoC容器的知识，然后介绍了依赖注入技术，最后又为大家说明了Bean作用域的各种配置。本章的重点是理解并争取掌握依赖注入的方法，难点是依赖注入关于注入其他Bean部分知识的理解。希望读者多加练习，力争为Spring技术的运用和学习打下坚实的基础。

第18章 面向切面编程

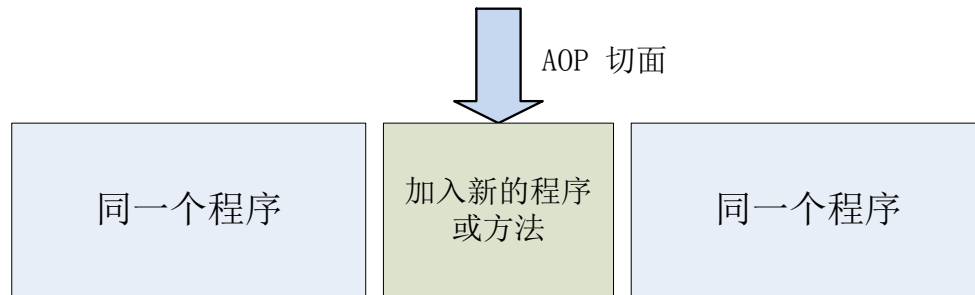
- * 本章继续学习Spring的第二大功能，即面向切面编程（Aspect Oriented Programming, AOP）。在前面学习Java时，我们知道Java是一门面向对象的语言。面向切面编程在一定程度上弥补了面向对象编程的不足。面向对象编程是对父类、子类这种纵向关系编程，而面向切面编程是在方法的前后进行横向关系编程。这一章我们就来对面向切面编程进行详细讲解。

18.1 面向切面编程简介

- * 对于初学者而言，面向切面编程是一个全新的编程思想。这里提出了切面的概念，通过切面对关注点进行模块化，其中最常见的切入点就是方法。使用面向对象编程，在指定方法的前面执行另一个方法是不难实现的，但是如果要在同一个包下所有类的方法前面都执行统一方法就是一件比较复杂的工作了。在这种情况下，如果使用面向切面编程就很容易实现。

18.1.1 面向切面编程的概念

- * 在面向切面编程中，经常会看到很多抽象的概念，就像我们刚开始学习Java时，也会遇到类、对象、多态等这些难懂的对象一样。我们先来对AOP中经常遇到的一些概念进行讲解。
- * 首先是切面，切面就像是一把刀，可以讲一个事物一分为二。Spring中的切面可以将一个程序分为两部分，在中间加入自己想做的事。而且可能不仅仅是对一个程序，也许是一个包下所有的程序，如图18.1所示。



18.1.1 面向切面编程的概念

- * 在一个程序中使用切面，通常是对程序中的方法进行的操作，方法调用和处理异常等待时间段称为连接点。在面向切面编程中，一个连接点就代表一个方法的执行。在Spring中，执行切面编程要用到拦截器的概念，当运行到某一切入点时，会有拦截器响应，这就是通知的概念。

18.1.2 面向切面编程的功能

- * Spring AOP是使用纯Java语言编写的，所以我们可以很容易的完成对它的扩展。在目前的Spring面向切面编程中，仅支持以方法作为连接点。之所以会这样，是因为在实际开发中以方法作为连接点是应用最多的，对于成员变量的操作是非常少的。下面我们就来重点学习Spring中使用方法做连接点的面向切面编程。

18.2 使用注解方式进行AOP开发

- * 注解是Java 5.0版本中提出的新特性技术，通过注解可以使一个普通的Java程序完成特定的功能。在Spring中，要使用注解方法进行AOP开发，需要用到AspectJ组件技术，通过使用AspectJ的库可以完成对切点的解析和匹配。

18.2.1 启动AspectJ的支持

- * 要想在Spring的面向切面开发中使用注解方式，就需要使用到AspectJ组件技术。要使用AspectJ技术首先要导入AspectJ相关的两个jar包：aspectjweaver.jar和aspectjrt.jar，它们位于lib/aspectj目录下。
- * AspectJ可以在其官方网站：<http://www.eclipse.org/aspectj/>进行下载。
- * 解压aspectj-1.7.1就可以得到需要的jar包，我们只要将其拷贝到创建的项目文件夹下就可以了。导入了AspectJ相关的jar包之外，我们还需要在Spring的applicationContext.xml配置文件中。

18.2.2 声明切面

- * 在启动AspectJ的支持之后，我们在Spring中开发Bean程序时，需要在Bean类的前面加入@AspectJ，从而标明该类是Spring的切面。我们举一个示例如图18.4所示。

```
@Aspect  
public class BooksServiceImpl{  
    //类主体  
}
```

在Bean类的前面加入@AspectJ，
标明是Spring的切面

- * 同样，开发完Bean之后，我们也要在Spring的配置文件中对其进行配置，配置方法如图18.5所示。

```
<bean id="booksServiceImpl" class="BooksServiceImpl">  
</bean>
```

18.2.3 声明切入点

- * 切入点决定了连接点关注的内容，使得我们可以控制通知什么时候执行。我们已经知道面向切面编程仅对方法执行，所以切入点也仅仅是判断哪些方法需要进行面向切面编程。
- * 声明切入点需要使用@Pointcut注解，并在后面给出切入点表达式，定义关注哪些方法的执行。最后还要给出一个切入点名称，它通过一个没有返回值的普通方法构成。我们可以给出一个例子，如图18.6所示。

```
@Pointcut ("execution(*dao..*.*(..))")  
  
private void allMethod() {  
  
}
```

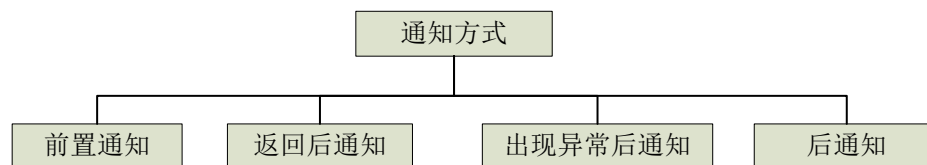
声明切入点的固定格式

切入点表达式

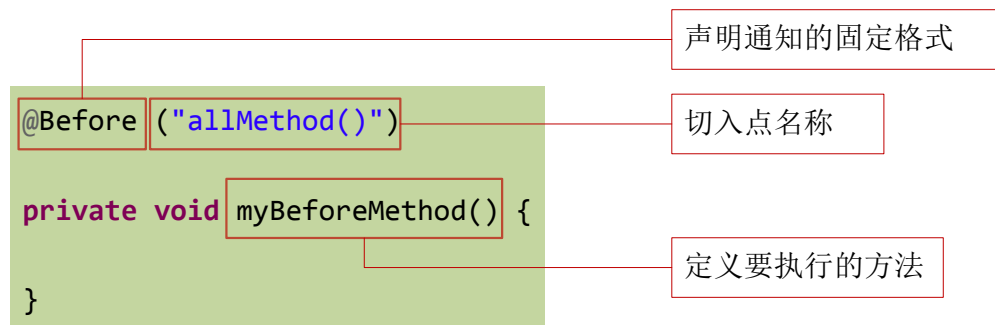
切入点名称

18.2.4 声明通知

- * 声明通知的作用是告诉程序应当在切入点的什么时候执行下面的方法。通知有四种类型，如图18.7所示。



- * 其中最常用的前置通知是指在切入点方法运行之前进行通知，从而执行下面的方法。前置通知使用的是@Before注解。我们可以给出一个例子，如图18.8所示。



18.3 使用注解对数据访问层进行管理

- * 我们对如何使用注解方式进行AOP开发已经有了一定的了解，现在我们就通过一个实际的例子来看怎样使用注解对数据访问层进行管理。
- * 在实际开发中，数据访问层是不可缺少的一部分。在数据访问层中通常要具有增、删、改、查等一系列方法。在开发过程中，有这样一个需求，不管进行任何操作时，都要通知程序员。如果我们不使用面向切面技术，就只能在每一个操作数据库方法前加入一段代码，这是非常重复的。我们现在就通过注解方式使用切面编程解决这样一个需求。

18.3 使用注解对数据访问层进行管理

- * 完成这样一个功能首先要定义数据访问层代码，然后定义一个使用注解的程序，在其中定义切面、切入点 and 通知。其中实现代码文件如表18.1所示。

文件名	文件功能
UserDAOImpl.java	数据访问层实现类，在其中定义了增删改查功能
AspectJAnnotation.java	面向切面编程类，在其中定义在进行数据库操作前执行的代码
applicationContext.xml	Spring的配置文件，在其中将数据访问层类和面向切面编程类进行配置
UserClient.java	用户功能测试程序，通过该程序来进行面向切面功能的测试

18.4 切入点

- * 在前面讲解注解方法进行AOP开发中，我们已经提到了切入点的使用。本节再来深入对其进行讲解，包括切入点指定者、合并切入点和切入点表达式等知识点。

18.4.1 切入点指定者

- * 在进行Spring面向切面编程时，支持在切入点表达式中使用多种AspectJ切入点指定者。其中，最常用的就是前面使用到的execution，通过它来匹配方法执行的连接点。其他较为常用的切入点指定者如表18.2所示。

切入点指定者	连接点说明
within	通过限定匹配特定类型确定连接点
this	通过指定类型的实例确定连接点
target	通过目标对象确定连接点
args	通过参数确定连接点

18.4.2 合并连接点

- * 在Java中有“&&”、“||”和“!” 3种逻辑运算符。在切入点表达式中也可以使用这3种逻辑运算符，其中最常用的就是“||”，它表示两边的方法都可以声明切入点。我们可以举一个例子如图18.15所示。

```
@Pointcut("execution(*dao.*(..)) || execution(*service.*(..))")
```

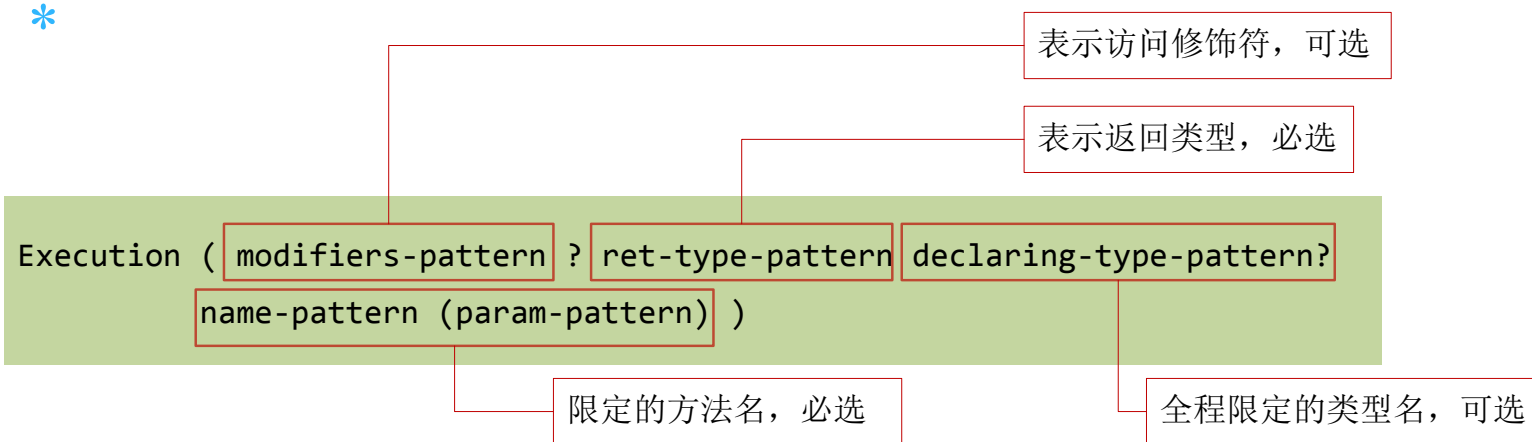
逻辑运算符“||”

- * 上面切入点表达式表示不管是dao包下的接口还是service包下的接口内的所有方法都会定义成切面的切入点，从而使它们都能够进行面向切面开发。

18.4.3 切入点表达式

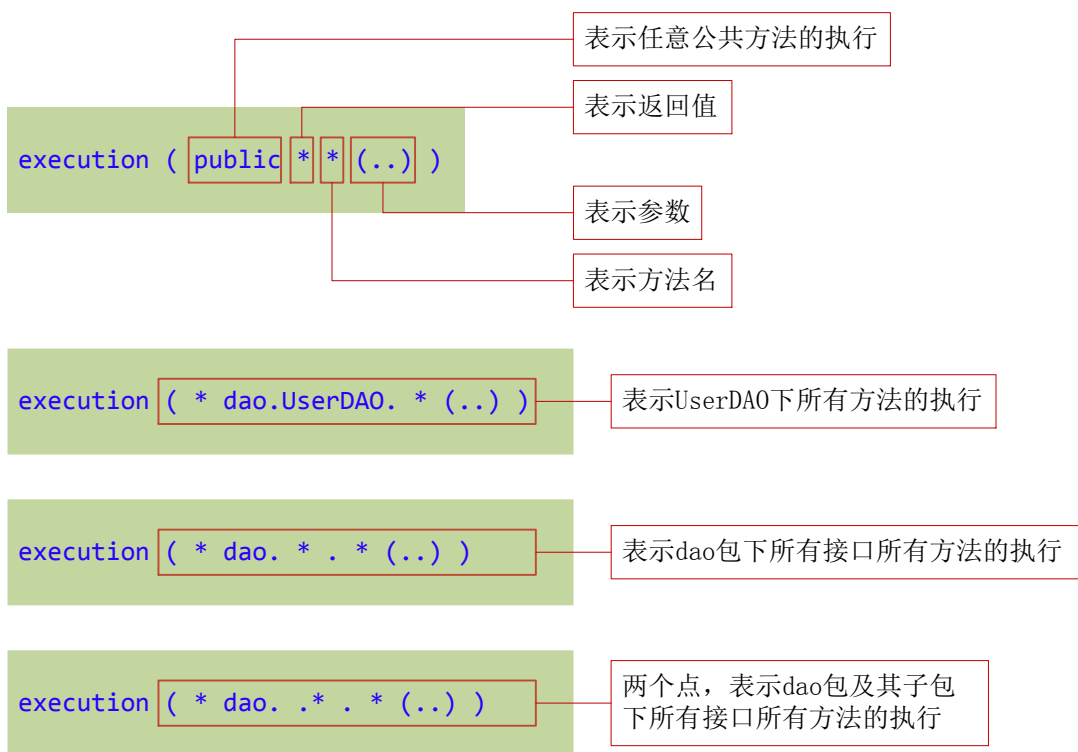
- * 读者可能对切入点表达式有很大的疑惑。本节就来通过execution切入点指定者来讲解切入点表达式，它的语法格式如图18.16所示。

*



18.4.3 切入点表达式

- * 我们再来列举几个简单的例子来帮助大家更好地理解切入点表达式，如图18.17所示。



18.5 通知

- * 在前面学习开发步骤时，我们也对通知有了一个简单的了解。通知是跟一个切入点表达式关联起来的，并且在切入点匹配的方法的某一运行时刻进行运行。除了前置通知外，我们对其他几种通知类型也进行进一步讲解。

18.5.1 返回后通知

- * 返回后通知是指当匹配的方法执行return语句返回值时进行通知，它使用“@AfterReturning”注解进行声明，其基本形式如图18.18所示。

```
@AfterReturning("allMethod()")
```

声明通知

```
public void myAfterReturningMethod(){  
}
```

定义通知方法

- * 注意：returning子句也是一种对匹配方法的限制，匹配方法的返回值类型必须要能转换为通知方法中的参数类型。

18.5.2 出错后通知

- * 出错后通知是指当执行匹配方法时出现异常后进行通知。它使用“@AfterThrowing”注解进行声明，其基本形式如图18.20所示。

```
@AfterThrowing("allMethod()")
```

声明通知

```
public void myAfterThrowingMethod(){  
}
```

定义通知方法

18.5.3 后通知

- * 后通知是指在匹配的方法结束后通知，它使用“@After”注解进行声明。需要注意的是，方法的结束有很多情况，例如正常运行结束、返回之后结束和发生异常，不管是哪种方式的结束都会进行后通知。后通知的基本形式如图18.22所示。

```
@After("allMethod()")
```

声明通知

```
public void myAfterMethod(){  
}
```

定义通知方法

18.5.4 环绕通知

- * 环绕通知是前置通知和后通知的结合体，使用环绕通知可以使一个方法的前后都执行通知方法。并且通过环绕通知可以决定什么方法什么时候执行、如何执行和是否执行。
- * 环绕通知是使用“@Around”注解声明的，在其中指定切入点名称。环绕通知的基本形式如图18.23所示。

```
@Around("allMethod()")
```

声明通知

```
public void myAfterMethod(ProceedingJoinPoint pjp){  
    Object retVal = pjp.proceed();  
    return retVal;  
}
```

定义通知方法

- * 注意：环绕通知并没有想象中那么好，它会使程序变得非常复杂。当只需要完成某一方面操作时，例如只是在方法运行前通知，最好还是使用前置通知。

18.6 在Spring中进行JDBC编程

- * 在Spring中可以整合多种数据库操作框架，例如整合Hibernate。本节我们先来讲解在Spring中如何进行最原始的JDBC编程。在JDBC编程中有很多重复的代码，如连接和关闭数据库链接等，在Spring中可以对这些重复代码进行封装，从而使程序员关注于业务的开发。

18.6.1 Spring中的数据库封装操作类和数据源接口

- * Spring中的数据库封装操作类是JdbcTemplate，它位于“org.springframework.jdbc.core”包下。使用该类可以完成数据库的连接和关闭，从而简化了JDBC操作。
- * 数据源接口也就是DataSource接口，通过使用该接口可以创建一个数据库连接。在Spring中有多种创建数据源的方式，本书主要通过Spring配置的方式建立数据源。我们知道，要想连接数据库最少要知道数据库的驱动、url、用户名和密码，所以在配置DataSource接口时需要给出这些信息。

18.6.2 创建数据库表操作

- * 学习完Spring中的数据库封装操作类和数据源接口，我们就可以使用其进行数据库开发了。我们先来看一个进行创建数据库开发的例子。

18.6.3 更新数据库操作

- * 更新数据库操作包括插入、修改和删除3种，这3种操作都是通过update方法完成的。在update方法方法中给出SQL语句参数，就可以完成相应的数据库操作。

18.6.4 查询数据库操作

- * 创建数据库表，并且向其中插入数据后，就可以通过查询操作将数据库表中的数据查询出来。在Spring的封装数据库中查询数据的方法有很多，这里我们使用其中的两个，分别是查询一个数据和查询所有数据。

18.7 小结

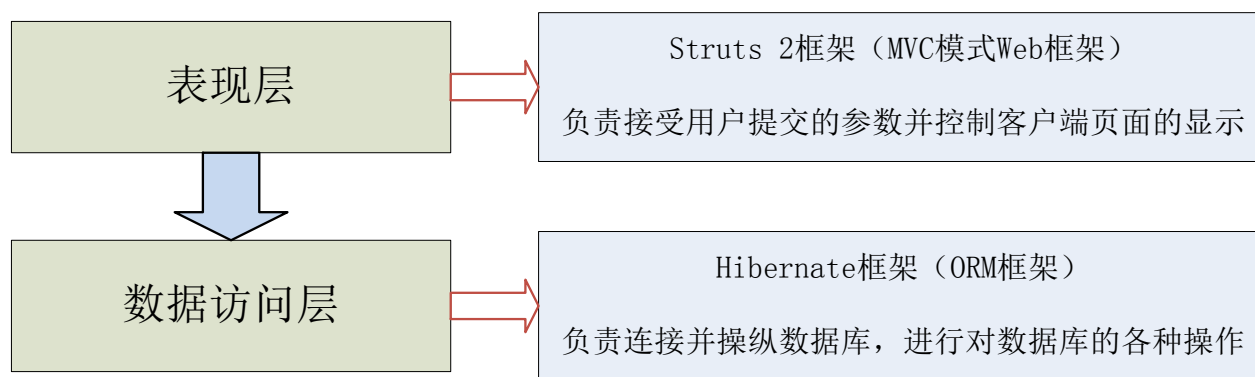
- * 本章讲解了Spring的第二个重要部分面向切面编程。首先我们为大家介绍了面向切面编程的相关知识，然后介绍了使用注解方式进行AOP开发的相关步骤，接着又为大家举例说明了如何使用注解对数据访问层进行管理，最后为大家讲解了在Spring中进行JDBC编程的操作。本章的重点是理解并争取掌握使用注解方式进行AOP开发和在Spring中进行JDBC编程的方法，难点是对切入点和通知等知识点的理解。希望读者多加练习，力争为Spring技术的运用和学习打下坚实的基础。

第19章 框架技术整合开发

- * 在前面的章节中，我们分别学习了Struts 2、Hibernate以及Spring三个框架。但是每个框架都是单独介绍的，并没有涉及整合的内容。在本章中我们将重点介绍3个框架的整合开发，包括Struts 2和Hibernate框架的整合开发、Struts 2和Spring框架的整合开发以及Hibernate和Spring框架的整合开发。通过本章的学习，读者应能够领会各框架的整合思想，从而迈进企业级开发的殿堂。

19.1 Struts 2和Hibernate框架的整合开发

- * 一个Web应用最重要的两部分，就是与用户交互的表现层和与数据库交互的数据访问层（DAO层）。Struts 2和Hibernate框架整合正好可以完美地实现这两部分的搭配，如图19.1所示。本节我们就来详细地介绍如何整合Struts 2和Hibernate框架。

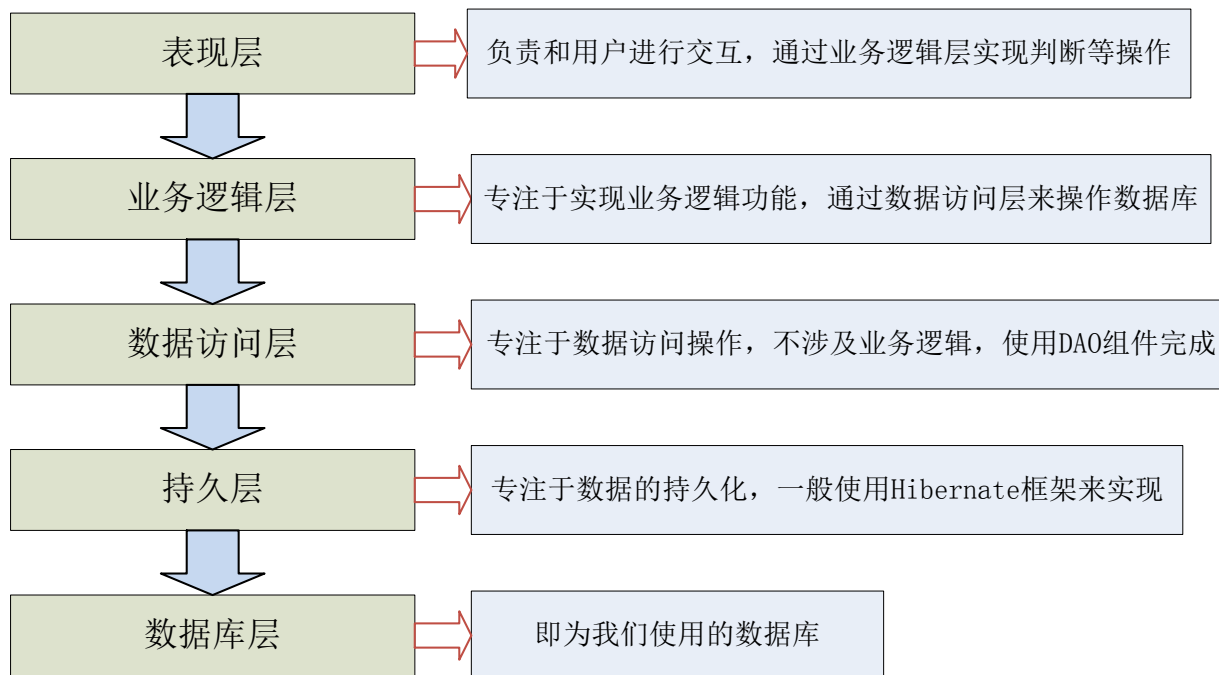


19.1.1 整合策略

- * 在整合Struts 2和Hibernate开发之前，首先我们必须清楚开发分层的思想。一个好的软件项目都是采用多层设计的，这就好比一个软件公司分为多个部门一样，每个层次负责不同的功能。在软件开发中使用分层的思想，就可以确定每层的工作任务，从而提高代码的内聚。

19.1.1 整合策略

- * 在Web开发中，一般采用5层架构，分别为表现层、业务逻辑层、数据访问层、持久层以及数据库层。它们各层的作用如图19.2所示。



19.1.1 整合策略

- * 注意：在实际开发中，我们有时候并不将数据库层放入Web开发中，而是将持久层作为底层。
- * 下面我们就通过一个网上书店的应用来演示Struts 2和Hibernate框架的整合策略。

19.1.2 数据库层开发

- * 数据库层用来接收数据访问层提交的数据，也供数据访问层获取数据。数据库层一般使用数据库管理系统，如MySQL数据库。在一个网上书店中要有一个用来存放书籍信息的数据库表，其中书籍信息包括书籍编号、书籍名称、书籍ISBN号以及书籍价格。

19.1.3 持久层开发

- * 持久层开发主要包含两个部分，一个是Hibernate配置文件hibernate.cfg.xml的开发，一个是实体类和映射文件的开发。关于hibernate.cfg.xml的开发我们不再赘述，请读者参考前面的相关章节。这里我们主要来开发实体类以及映射文件。
- * 创建了book表，就应该添加一个实体类Book来与之相对应。同时还需要添加实体类的映射文件Book.hbm.xml，从而完成Book类的对象关系映射。
- * 注意：配置完Book.hbm.xml后一定不要忘记在hibernate.cfg.xml添加进行相应的配置信息。即增加<mapping resource="po/Book.hbm.xml" />语句。

19.1.4 数据访问层开发

- * 数据访问层又称为DAO层，在该层中包含了所有的操作数据的方法，如保存数据、删除数据、修改数据和查询数据等。数据访问层包括3个组成部分，如图19.6所示。

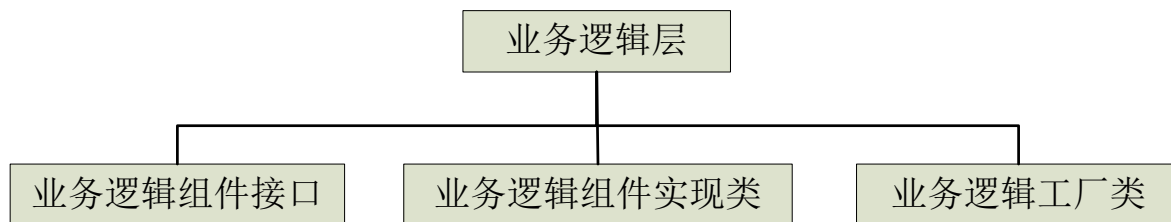


19.1.4 数据访问层开发

- * 首先我们来定义一个DAO接口BookDAO，在该接口中定义3个方法，分别用来添加书籍、根据ISBN号查询书籍以及查询所有书籍。
- * 注意：数据访问层是不包含任何业务逻辑的，所以在包含书籍时并不会查询该书籍是否存在。

19.1.5 业务逻辑层开发

- * 业务逻辑层的开发和数据访问层基本类似，不同的是数据访问层是通过Hibernate来完成数据操作，而业务逻辑层重点实现的是业务逻辑。业务逻辑层中的数据操作都是通过调用数据访问层来实现的。业务逻辑层包含3个部分，如图19.10所示。



- * 注意：业务逻辑层是这个Web架构中最重要的一部分，它是连接表现层和数据访问层的桥梁。

19.1.6 完成书籍的录入

- * 在5层架构模型的最上层是表现层，表现层一般使用MVC框架来充当，比如Struts 2框架。要完成书籍的录入，首先要添加一个书籍录入表单，用来接收书籍信息，然后需要添加一个书籍录入控制器，并调用业务逻辑层来完成书籍的录入。

19.1.7 完成所有图书的显示

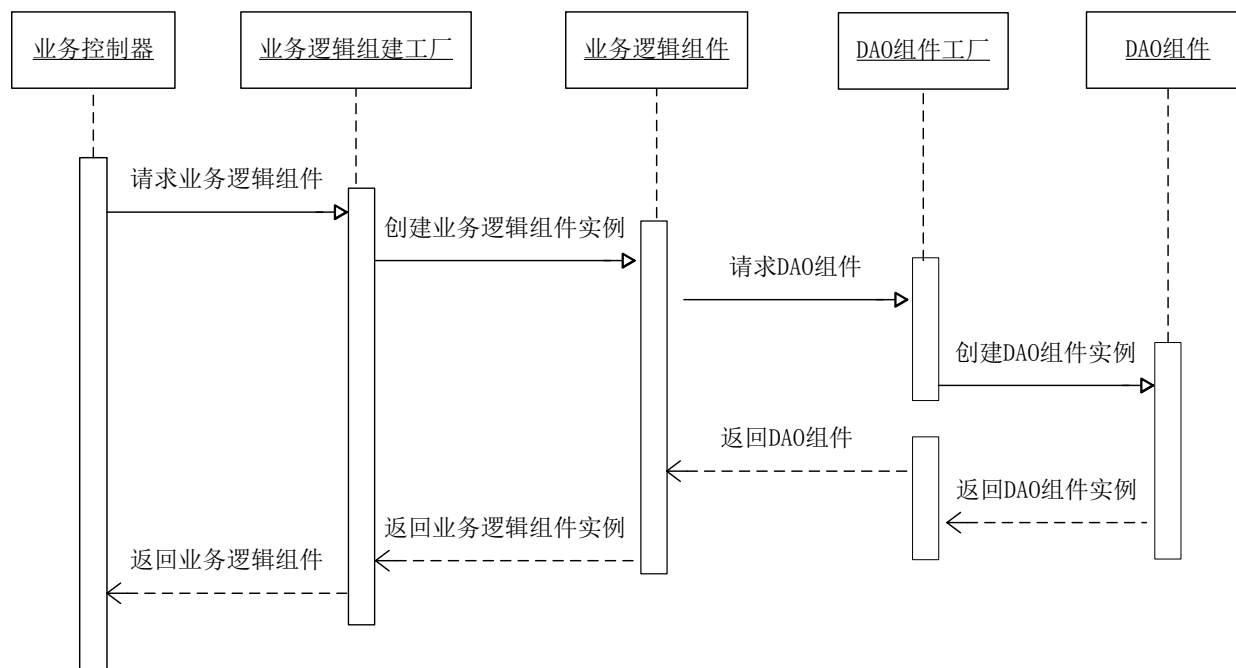
- * 要完成所有图书信息的显示，首先需要添加一个业务控制器来获得所有的图书，然后再添加一个图书列表显示页面，在该页面中循环显示所有的图书。

19.2 Struts 2和Spring整合开发

- * Spring本身提供了一个MVC框架，但是因为这套框架大量应用了映射策略，使得开发起来非常繁琐。Struts 2框架是一个非常优秀的MVC框架，这时可以通过Struts 2和Spring的整合，充分利用Spring的IoC特性，从而大大降低系统各层之间的耦合度。

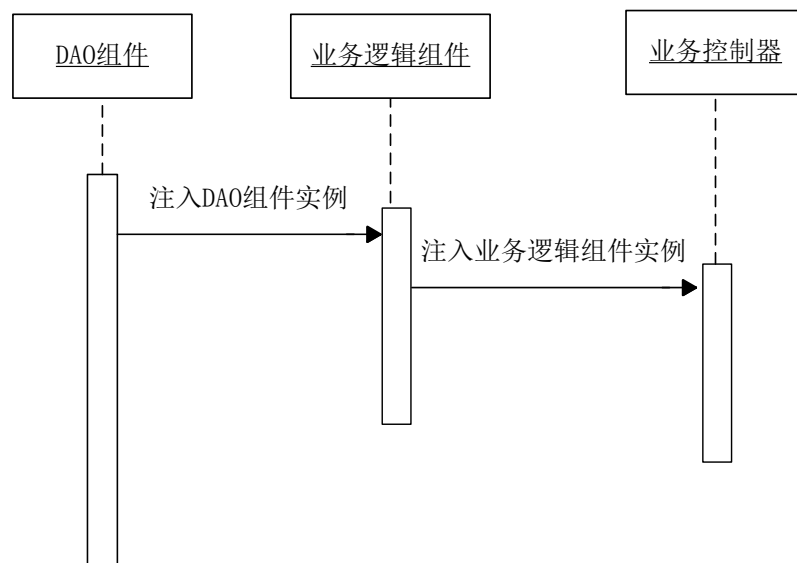
19.2.1 整合策略

- * 在使用Spring框架之前，各层之间通过使用工厂类来创建组件实例。工厂模式顺序如图19.23所示。



19.2.1 整合策略

- * 使用工厂模式确实可以大大降低各层之间的耦合度，但是同样也带来了代码编写的巨大困难。Spring框架的出现，很好地解决了这个难题。在项目中整合Spring框架，可以使用Spring的IoC容器来管理控制器，并通过依赖注入的方式为控制器注入业务逻辑组件实例。依赖注入顺序如图19.24所示。



19.2.2 安装Spring插件完成整合

- * 要整合Struts 2和Spring框架，需要两个条件，第一个是为项目添加Spring框架支持，第二个是安装Spring插件。
- * 首先我们来为项目添加Spring框架支持。第一步要在Web应用的WEB-INF\lib目录下添加Spring所需的库文件spring-core-3.2.0.M2。所以我们要先在Spring的官方下载网站 (<http://www.springsource.org/download/>) 下载Spring。在解压的libs包中找到spring-core-3.2.0.M2文件将其添加到我们所建项目的WEB-INF\lib文件夹下。
- * 第二步是修改Web的配置文件web.xml，我们在其中添加一个ContextLoaderListener监听器。
- * 通过添加该配置器，使得Web应用启动时会自动查找WEB-INF目录下的applicationContext.xml配置文件，并根据该配置文件来创建Spring容器。
- * 为项目添加完Spring支持后，还需要为Struts 2安装Spring插件，从而完成Struts 2和Spring的整合。安装Spring插件非常简单，只需将struts-2.3.4.1/lib目录下的struts2-spring-plugin-2.3.4.1.jar复制到WEB-INF/lib目录下即可。

19.2.3 装配数据访问层

- * 为项目添加了Spring框架支持后，就可以通过Spring来装配数据访问层，从而使得数据访问层由Spring的IoC容器来管理。

19.2.4 装配业务逻辑层

- * 在装配业务逻辑层之前，首先需要修改业务逻辑组件实现类，在该类添加一个数据访问层接口的类型，并为属性添加get和set方法。这样在装配业务逻辑层时，就可以在业务逻辑层中注入数据访问层。

19.2.5 装配业务控制器

- * 业务控制是用来直接跟用户进行交互的，这时需要调用业务逻辑层的方法从而完成特定的业务逻辑。业务控制器同样需要在Spring中进行装配，并为其注入业务逻辑层实例对象。

19.3 Hibernate和Spring整合开发

- * 前面已经讲了Struts 2和Hibernate、Spring这两个框架的整合，同样Hibernate和Spring框架也能进行整合开发。整合后就能通过Spring来管理Hibernate连接数据库的数据源，还能管理SessionFactory。在Spring框架中还提供了HibernateTemplate类和HibernateDaoSupport类来更方便地进行Hibernate操作。

19.3.1 使用Spring管理数据源

- * 在整合前，我们需要在Hibernate的配置文件hibernate.cfg.xml中对数据源进行配置。然后在整合后。再在Spring的配置文件applicationContext.xml中进行配置，我们就可以使用Spring容器统一管理数据源了。

19.3.2 使用Spring管理 SessionFactory

- * 整合前我们都是使用HibernateSessionFactory来负责创建SessionFactory，整合后就可以通过Spring来配置并管理SessionFactory了。通过装配的SessionFactory，就能够为其他DAO组件的持久操作提供支持。

19.3.3 使用HibernateTemplate类

- * 在Spring中提供了一个用于简化Hibernate操作的模板类HibernateTemplate。HibernateTemplate类中提供了一些简单的注入find、load或者delete操作的方法，以及可选择的快捷函数来替换回调的实现。

19.3.4 使用HibernateDaoSupport 类

- * 在Spring中提供了一个用于简化DAO开发的类HibernateDaoSupport。该类提供了一个setSessionFactory方法来接受一个SessionFactory对象，从而完成SessionFactory的注入。还提供了一个setHibernateTemplate方法来接受一个HibernateTemplate对象，从而完成HibernateTemplate的注入。同样，HibernateDaoSupport类也提供了getSessionFactory和getHibernateTemplate方法来获得SessionFactory对象和HibernateTemplate对象。

19.3.5 使用Spring管理事务管理器

- * 在TransactionTemplate中必须包含一个PlatformTransactionManager实例对象。该实例可以通过代码来创建，也可以使用Spring来进行依赖注入。不管怎样，只有获得了PlatformTransactionManager的引用，TransactionTemplate才能完成事务操作。

19.3.6 为业务逻辑层注入事务管理器

- * 在Spring中创建完成事务管理器后，需要在业务逻辑层中添加一个属性，通过该属性来接收事务管理器。

19.3.7 使用TransactionTemplate 进行事务管理

- * 完成事务管理器的注入后，就可以通过该事务管理器来实例化一个TransactionTemplate。他通过调用TransactionTemplate的execute方法就可以进行事务操作。

19.4 小结

- * 本章是Struts 2、Hibernate以及Spring三个框架的整合开发章节，我们依次实现了Struts 2和Hibernate框架的整合开发、Struts 2和Spring整合开发以及Hibernate和Spring整合开发。本章的重点和难点都是能否完成整合框架的开发配置，并实现不同框架的整合。通过本章的学习，希望读者能够领会各框架的整合思想，以便今后能够更好地实现具体的企业及开发。

電子工業出版社